PS ARCHIVE
966
OHNSON, J.

METHODS FOR DIGITAL SIMULATION OF MILITARY CONFLICT SITUATIONS

JAMES E. JOHNSON

U.S. NAVAL P[illegible]
MONTEREY, CALIFORNIA

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

METHODS FOR DIGITAL SIMULATION OF MILITARY CONFLICT SITUATIONS

* * * * *

James E. Johnson

METHODS FOR DIGITAL SIMULATION OF MILITARY CONFLICT SITUATIONS

by

James E. Johnson
Lieutenant, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

United States Naval Postgraduate School
Monterey, California

1 9 6 5

NPS ARCHIVE
1966
JOHNSON, J.

Thesis
J62

Library
U. S. Naval Postgraduate School
Monterey, California

METHODS FOR DIGITAL SIMULATION OF MILITARY CONFLICT SITUATIONS

by

James E. Johnson

This work is accepted as fulfilling
the thesis requirements for the degree of
MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS
from the
United States Naval Postgraduate School

ABSTRACT

Digital computers are used in several ways to simulate military situations. Three types of these simulations are examined: an off-line simulation using a special simulation language (MILITRAN), a very similar off-line simulation using a general purpose language (FORTRAN 63), and an on-line simulation using FORTRAN 60 and FORTRAN Symbolic. The essential features of a simulation language are identified. These features lead to the identification of a class of problems for which the simulation language is superior to the general purpose language. A sample problem is coded in both MILITRAN and FORTRAN 63. An appendix contains a description of MILITRAN in a format allowing a convenient comparison of features with other languages available at USNPGS.

An on-line simulation is presented It employs and demonstrates some possibilities of the satellite mode of the CDC-1604. On-line simulations are seen to possess advantages over off-line simulations for a large class of users. Areas where these advantages should be more fully exploited are noted. An appendix contains detailed instructions for writing an on-line simulation for the USNPGS satellite system.

The portion of this work involving MILITRAN was accomplished while the author was on temporary duty with ONR in Washington, D. C., during the summer of 1964. The remainder of the work was accomplished at USNPGS.

TABLE OF CONTENTS

Section	Title	Page
1.	Introduction	1
2.	The Problem	2
3.	Off-line Simulations	3
4.	On-line Simulations	16
5.	Conclusions	20

Appendix

I.	Description of MILITRAN	28
II.	Off-line Simulation Programs and Flow Charts	41
	MILITRAN	45
	FORTRAN 63	64
III.	Instructions for Writing On-line Simulations	83
IV.	On-line Simulation Program and Flow Charts	89

LIST OF ILLUSTRATIONS

Figure		Page
1.	Off-line simulation playing area	41
2.	Array LCLASS	43
3.	Array TRANSC	43
4.	Sample of off-line simulation output	44
5.	On-line simulation playing area	89
6.	DD-65 scope presentation to on-line player	90

1. Introduction.

The simulation of military conflict situations is a useful and frequently employed tool for both the operations analyst and the line officer of the Armed Forces. Simulation enjoys a current popularity, but it is in no sense new. Many authors have recorded how the early Chinese devised "war chess" to teach their young noblemen some of the concepts of battle without the danger of bloodshed. We also know that the Prussians, French, and later the Nazis made very effective use of a type of war game played on maps. All of those who seriously employed this type of simulation felt it was of benefit to the officers.

This means of gaining information and sharpening the mind, with no ill effects from bad decisions, has certainly not been limited to the military. Inventors and scientists have always used models of one kind or another to aid their thinking.

Despite this long history, the field of simulation took on new importance and utility as digital computers became available. Problems previously too complicated for modeling were attacked. Big city traffic congestion, global air battles, and even the newly conceived logic structures of future computers are now simulated.

The new possibilities of simulation stimulated new interest and much new growth in the field. To make it easier to write computer programs for these simulations new languages were developed. A few of these include SIMSCRIPT, GPSS, CSL, SIMTRAN, DEPI, DYNA, BLODI, DYSAC, PARTNER, COBLOC,FORBLOC, HYBLOC, MADBLOC, SIMPAC, CLP, GASP, SIMPLE, DYNAMO, DAS, JSS, SOL, and SILLY. One language, MILITRAN, was devised

primarily for military simulations.

2. The Problem.

This rapid growth in the field of simulation is taking place at the same time that a similar growth is taking place in the entire computer field. During the years which it has taken to develop the simulation languages, new and better general purpose languages have been developed. In view of this parallel development, it is well to examine the languages which have been produced. To properly do this it is necessary to re-examine the essential features of a simulation language.

When much of the current interest in simulating military conflict situations began, the digital computer was available only in computer centers at universities, in industry, and at large military installations. Now the digital computer is a common item aboard modern war ships. Hardware advances in the fields of time sharing and display have made the computer more available and versatile. The idea of several users having access to the same central computer from their own separate offices is a reality.

These developments suggest some questions. Have better general purpose languages removed the need for simulation languages? Due to the hardware advances are there new applications of simulation that we should employ? Is it possible that some of these new applications offer particular advantages to the military?

To state that the total answers to these questions were reported

here would be presumptious. One viewpoint and one examination can only be expected to represent a part of an answer. With this qualification implied, this study has indicated some conclusions.

Through a comparative study of a specific simulation language, a tentative answer to the first question emerged. It appears that for certain simulation applications the general purpose language has indeed removed the need for a simulation language. At the very least, the study has shown that the prospective user of a simulation language should be cautious. Since the user will pay a stiff price for the language he must know exactly what he expects from the language--and he should insure the language fills these needs.

Comparing off-line simulations with on-line simulations shows the two to be complementary rather than competitive. Within the military the on-line simulation is seen to be relatively under exploited, although display advances have made it more realistic and computer availability has made it more widely available. Finally, and this without qualification, a Navy need for expanded on-line simulation is shown along with a latent capability to provide the simulation with installed equipment.

3. Off-Line Simulations.

An examination of possible computer simulation techniques indicates these can be broadly classed as either off-line or on-line simulations. The off-line simulation of a military conflict situation is normally of the Monte Carlo variety. Program decisions are based on some preassigned

probability. For example, if a target detection is involved it is assigned a probability of occurrence--say 70%. A random number is then generated from a uniform population between 0 and 1.0. If the number produced is less than 0.7 the play of the game continues on the basis of a detection having been made. If the number is greater than 0.7 the play of the game continues as if no detection had occurred. All similar alternate decisions are handled in the same way.

To rule out the possibility of a very atypical selection of random numbers coloring the outcome of the game, it is replayed many times with the same parameters but with a different population of the random numbers from the same distribution. The average result of these many plays of the game is then the expected outcome. When this outcome is identified, it is possible to change some single parameter, corresponding perhaps to a proposed change in the disposition of forces or a change in their capabilities. The simulation may then be re-run until a new expected outcome is determined. Thus, the effect of new tactics or a new weapons system may be evaluated. This, although certainly an oversimplification, captures the flavor of the off-line simulation.

These off-line simulations have been written in machine language or assembly language, in general purpose languages such as FORTRAN or ALGOL, and of course in the special simulation languages such as SIMSCRIPT or MILITRAN. To learn what advantages a simulation language offered over a general purpose language, a small scale simulation was coded in MILITRAN and then the same problem was coded in FORTRAN 63.

The MILITRAN program and the FORTRAN 63 program are contained in Appendix II. A description of the MILITRAN language is in Appendix I.

Studying the problem in this way could have biased the result. A comparison of a simulation language with a general purpose language will only develop if FORTRAN 63 is a typical general purpose language and MILITRAN is a typical simulation language. In any strict sense no language is really typical of either of these classes. In a broader way they are both good representatives. FORTRAN 63 and its close relative FORTRAN IV qualify through usage and availability. MILITRAN is certainly one of the newer simulation languages to become available. It was submitted to the Office of Naval Research for a pilot study in the summer of 1964. This author performed that study. The MILITRAN manuals were released as the study was concluding. A study of other simulation language publications was made to determine if MILITRAN lacked any of the typical characteristics. None were noted. MILITRAN is therefore considered a fair representative of current members of its class.

What advantages did the coding of a simulation in a special simulation language offer over the coding of the same program in a general purpose language? This question implies another. A simulation language is really two languages. An underlying general purpose language exists. On top of this language are constructed some simulation features. These features go together with the general purpose language to form the

simulation language. Thus we seek, in a comparison of a general purpose language and a simulation language, to separate any advantages the simulation language may have due to a superior underlying general purpose language from those advantages due only to its simulation characteristics. It is necessary to address ourselves to a new question. What features of the simulation language under study, MILITRAN, are a part of that language strictly for simulation purposes?

The segregation of features is an arbitrary task, but writing the simulation in both languages caused a few characteristics to stand clearly apart.

Although most activity in the real world is continuous, the digital computer only operates in a discrete manner. To simulate the continuous activity it is necessary to break it down into discrete events. The flow chart on page 45 shows such a breakdown. Here the continuous advance of a transiting submarine is simulated by a discrete event wherein a step of motion takes place. Similarly, an attack upon the transitor is handled as a discrete event. These events are termed contingent events. The block of coding which assesses the condition of the participants (transitor destroyed? transit complete? etc.) is a permanent event.

Certainly one unique feature of the simulation language is its ability to properly sequence the execution of these events. MILITRAN provides for permanent events and contingent events by that name. It brings about a sequencing of the events through a variable--TIME.

Every contingent event automatically possesses a variable called TIME which can be set and reset by a line of coding any place within the program. After a pass through the execution of all permanent events, a decision is made as to which, if any, contingent event will be executed. The criterion is to execute the event containing the smallest value of TIME which is larger than the current TIME. Thus if current TIME were 10.0 and contingent events contained the following times--10.2, 11.0, 9.9, and 10.0--the event containing TIME=10.2 would be executed.

Upon execution current TIME would be advanced to 10.2. After the contingent event had been completed, all permanent events would again be executed. These might or might not change the value of TIME in the contingent events. Upon completion of the permanent events a new decision would be made concerning the contingent event to execute.

A general purpose language can accomplish the same objective in several ways, but as the complexity of the problem grows, so does the programming difficulty. In simple cases a "computed go to" suffices. This is employed in the example program. See page 64. As long as the desired program jump is the one most recently designated this works well. If a hierarchy of importance exists MILITRAN simply allows the increasingly higher priority events to be planted with times at increasingly smaller increments in the future.

Another way to duplicate this with a general purpose language is to create a priority list and then check this list to find the event of highest priority which is tagged for execution. The complexity

arises if certain circumstances cause priorities to change as the program progresses. Perhaps the very best way to schedule events is to duplicate MILITRAN'S time system with a time array.

Another of the clear simulation features of MILITRAN is that it gives the programmer a new variable to use. This is the OBJECT. Real variables, complex variables, integer variables, and logical variables, even in combination, are insufficient to completely describe the real world. Some simulations by their nature involve objects. How these objects act and change and how they are acted upon is central to these simulations. MILITRAN allows for this by having available OBJECTS and PROGRAM OBJECTS. The PROGRAM OBJECT is a variable which may take on the value of one of several OBJECTS. For example, a target might be defined as a PROGRAM OBJECT. As the program progressed, the target might alternately become the OBJECTS tank, rifle platoon, aircraft, and machine gun nest. As the target changed so would the required tactics.

We can assume that anything which we wish to model on a computer can be satisfactorily described by a state vector. Some dimensions of this vector may appear in the real world as best described by OBJECTS. Since we do try to model on the digital computer we imply that each dimension has only a finite number of possible values of interest. If the simulation is to be written in a general purpose language, we are perfectly free to assign a number to each of the objects and allow a real variable to take on the value of these numbers just as the PROGRAM OBJECT takes on the values of the several OBJECTS. Although this

removes an exterior similarity to the real world and makes the program less readable, it in no way limits program accuracy nor does it add undue complication to program writing. If the external similarity is desired, the general purpose languages permit a variable to be set equal to a group of Hollerith characters. It is possible then to realize the same capability with a general purpose language, but at a cost of additional programming difficulty depending on our specific needs.

The CLASS declaration also contributes to the simulation ability of MILITRAN. This feature makes it possible to group objects according to some common characteristic. Once these groupings have been created it is possible to make decisions based upon group membership. The classes may overlap and their membership may change as the program progresses.

To duplicate this action when using a general purpose language it is convenient to resort to a logical array. Each object may be assigned a row in the array, and each possible class may be represented by a column. Class membership is then indicated by a logical true in the appropriate row and column See Figure 2, page 43. This allows the program to accomplish the same ends as the MILITRAN program, but there is a significant loss of convenience and ease of programming.

Simulations seem to lead to the use of lists. These lists change as the program progresses and often the processing that takes place depends upon the contents of the lists. In addition to checking the

contents of a list it is sometimes convenient to know the number of items in a list. MILITRAN is well equipped to service these needs. Built into the MILITRAN language are instructions which cause additions, replacement in, or deletions from lists. Searching a list for the first member, the last member, or a random member which fulfills certain requirements is permitted. The length of the list is automatically maintained and updated for ready reference. Each of the entries in a list may consist of several elements which are required to describe the entry. It is possible for these descriptive elements to possess different modes. For example, one list could contain entries for each of the ships in a particular battle. Within the list each entry could have associated elements of logical, integer, real, and program object modes as required to fully describe the particular ship. It is of course possible to alter individual elements within the list as well as to process the entire entry.

A general purpose language allows this structure to be approximated. Several arrays may be required to completely describe the members of one list. A logical array, an integer array, and a real array may all be used. If each of these arrays has a common index the fact that they are different arrays is only a matter of definition. By writing his own iterative procedures the programmer can accomplish any of the list processing built into MILITRAN. This, although possible, is in no way as convenient as it would be with MILITRAN.

Although this paper has, so far, purposely tried to exclude from consideration any of the details or advantages of the general purpose

language underlying MILITRAN, this attempt can produce some difficulties. It is essential not to confuse the general purpose features of a language with its special purpose features. It is also essential to recognize the vital nature of this underlying general purpose language. Because it does give the simulation language its life, a few of the features will be briefly mentioned.

MILITRAN allows statement labels to have as many as eleven alphanumeric characters. Identifiers of variables may have as many as sixty characters. The program may therefore consist entirely of meaningful names and is a clearer documentation of itself. The compiler accepts a large amount of flexibility. In general, if more than one unambiguous form exists, all of them will be acceptable. This flexibility extends to several areas. For example, the programmer may declare the mode of each of his variables, or he may build rules identical to those of FORTRAN so that the first letter of variable identifier automatically determines the mode, or he may set up some mixture of the above two extremes that most suits his individual problem. A step toward more complete error detection on a single computer run and more meaningful error reporting has been taken by MILITRAN.

Both MILITRAN and FORTRAN 63 contain some general purpose features which are quite useful in simulations, but which were not available in certain of the earlier languages. Probably the most useful of these is the ability to handle complex Boolean expressions with a single line of coding. This is facilitated by recognizing such logical operators as NOT, AND, and OR.

In addition to differences noticed from the programmer's point of view, the computer run uncovered some other characteristics of the simulation language.

The MILITRAN program was run on an IBM 7094 model II, while the FORTRAN 63 program was run on the CDC 1604. Any valid comparison of languages involving compilation, assembly, or running times should of course be made with the same type of computer. Since MILITRAN is not presently available for the 1604 and since no 7094 was available at the Naval Postgraduate School this was not possible. Even accounting for the uncertainties involved in this type of comparison, some general results will be indicated.

The two problems are as similar as is possible. This even extends to leaving known inefficiencies in the FORTRAN program just to maintain the similarity.

The figures in Table 1 resulted from the test.

	MILITRAN (FAP)	FORTRAN 63 (CODAP-1)
Source language lines of coding in main program exclusive of declarations.	167	196
Source language lines of coding devoted to declarations.	39	9
Object language instructions produced upon assembly.	1601	854
Adjusted object language instructions for certain miscellaneous tasks. (See discussion below.)	319	278
Object language instructions for basic simulation	1193	548

	MILITRAN (FAP)	FORTRAN 63 (CODAP-1)
Total memory addresses occupied by main program and reserved common storage. (octal)	4766	2423
Main program plus subroutine compilation and assembly. (in min.)*	3.76	1.96
Main program compilation and assembly. (in min.)*	2.55	1.58
Execution time. (in min.)	0.22	0.24

*tape motion time is included in these figures

TABLE 1.

DATA FROM SAMPLE PROGRAM RUNS

The miscellaneous tasks mentioned in the above figures included setting up arrays, assigning variables to common storage, coding the BCD characters for the initial data printout, providing for a typewriter message to the operator and other incidentals prior to starting the iterative part of the simulation. Due to a difference in the way FAP and CODAP-1 record BCD information, the FAP program uses 85 BCI instructions to code certain output statements, while CODAP-1 required only 28 separate BCD statements but used many intervening cells to pack the same information. The figures for these miscellaneous operations in the table have been adjusted by subtracting the BCD and BCI instructions from their respective columns.

The reader will be aware that in general the 1604 address holds two instructions. This accounts for some of the difference in memory

occupancy. Both programs use very nearly the same amount of reserved storage. MILITRAN used 1216 cells while FORTRAN used 1200 cells for the same storage. Thus the MILITRAN program proper uses 3550 cells while the FORTRAN program occupies 1223 cells. One means of trying to cancel out the effect of the two 1604 instructions per computer word is to add a figure equal to one half of the CODAP-1 instructions to actual memory occupied by the assembled FORTRAN program. This yields at most 2070 pseudo cells for FORTRAN and puts MILITRAN at a 1.75 to 1.0 disadvantage on memory occupancy. (Throughout the above discussion all references to memory cells are octal.)

It is beyond the scope of this paper to derive a valid figure for an exact speed comparison of the 7094 model II and the 1604, but whether the 7094 is four times faster or ten times faster the conclusions are the same. A MILITRAN program requires significantly more computer time than the FORTRAN 63 program.

This is a general and uncontested complaint about simulation languages, and no one reasonably expects it to be otherwise. Every computer language purchases programming ease at the expense of computer time. Each user must decide if the programming ease is worth the price. Using the lower figure of four to one for the difference in computer speeds lets MILITRAN benefit from the unresolved doubt. Even at this, MILITRAN takes four times longer to run and eight times as long for compilation and assembly.

Before drawing any conclusions, one other factor should be mentioned. MILITRAN offers a noticeably more extensive first run error detecting

and reporting ability. This, and not any of the simulation features, accounts for some part of the speed disadvantage during compilation and assembly. It also tends to pay for itself. In this test case the FORTRAN 63 program required two and a half times as many debugging runs as the MILITRAN program. This was due, at least in part, to the superior quality and quantity of the diagnostics.

Writing a simulation in the two different languages has led to these general observations. The simulation language offered clear, but limited, advantages to those who program simulations. It made it possible to write a more readable simulation with more exterior similarity to the real world it attempted to model. It required relatively little imagination to program. In the small scale example there was no difference in the total number of lines the programmer had to write. The specific simulation language used for the test offered clearer and more complete diagnostics on the first run. The advantages mentioned were gained at the expense of learning a special language, obtaining a significantly larger object program and expending considerably more computer time.

Who, then, does this study identify as prospective users of the special purpose simulation language? The class is seriously restricted. Those who will benefit most from simulation languages will deal primarily with long simulations--significantly longer than the example problem. Their problems will benefit most if they have a large, complicated event structure, and have a large dependence on lists. Yet, simultaneously the prospective user must be relatively unconcerned about

the size of his assembled program in memory or about the running efficiency. This is indeed a restricted class.

Note how many who frequently write simulations are excluded. The authors of the simulation languages never intended this. Nor, did they fail to produce as good a language as they intended. It seems likely that the authors of the various simulation languages first attempted to write simulations in general purpose languages. Their difficulties revealed inadequacies of the available languages. There were sufficient difficulties for many of the users to decide that it was worthwhile to write a new language to better serve them.

Others who were working on general problems must have experienced some of the same difficulties, because the current general purpose languages are better vehicles for simulations than their forerunners. (FORTRAN 63 is a better language to use for simulations than FORTRAN 60.) This generates a suspicion. Are simulations by their very nature so unique that they require a special purpose language? The answer to that question will be deferred until simulations in general have been more fully examined.

4. On-Line Simulations.

Simulations may be classed as to the number of participators with conflicting interests. In the off-line simulation the decisions as to course of action of all the conflicting participants are determined either by fixed rules or with the aid of random numbers. An on-line simulation makes it possible for the courses of action of one or more of the participants to be determined by a human player or players. This type of

a simulation is accurately termed a game. If only one human player is involved the part of his opponent is played by the computer with its fixed rules and random numbers. If more than one human player is permitted the humans play against each other and the computer acts as their source of information, their means of communication and an impartial referee.

To gain better appreciation of the advantages and a clearer understanding of the problems, an on-line simulation was written employing the satellite features of the CDC 160-1604 and DD 65 system. [5., 11.] FORTRAN 60 and FORTRAN Symbolic were used for the simulation. The problem is designed for one player due to equipment limitations, but could be modified for two players. The player is presented target information via typewriter messages and the DD 65 graphical display. He is periodically given an opportunity to maneuver his ship so as to arrive in position for a torpedo attack. He does this through the electric typewriter on the DD 65. The problem being simulated is a submarine approach using only passive sonar information, so the player is given no range information. When he believes his ship is in firing position he may launch an attack. The computer assesses the attack and initiates appropriate action. The problem ends when the submarine sinks his target, is sunk by the target, or the target escapes the submarine. At the completion of each problem a critique is held by the computer. The target track and own ship track are displayed. The previously unknown target data is supplied and the results of the transit are assessed.

The example is extensive enough to provide a fair degree of realism, but it in no way taxes the computing equipment. The program, its common storage and all the required subroutines occupy only 7200 octal addresses in memory. Appendix III contains detailed instructions for writing an on-line simulation for the USNPGS satellite system, and Appendix IV contains the example program.

More than the external differences separate the on-line simulations from off-line simulations. Their purpose and the class of user they serve vary. The off-line simulation provides an output of data and statistics. A study of these can lead to a determination of an optimum employment of forces or equipment. An operations analyst can and does employ this type of simulation as a part of his scientific study of some aspect of warfare. Off-line simulations are most useful as an aid to planning.

The on-line simulation might be quite useful without producing any hard copy output. It can provide a form of decision making practice. By presenting the user with a certain amount of data, allowing him to make a decision in the face of that data, and then assessing the implications of that decision; the on-line simulation can serve the user. Statistical results of the runs are not necessarily maintained. In this event the on-line simulation is not an analytical tool as is its off-line brother. It is little more than a training device. This does not mean that it is any less useful.

Those responsible for the conduct of warfare in the face of the enemy could use both types of simulations. This military line officer

together with his operations analyst would employ off-line simulations and many other tools to help build the best possible plans. Even conceding that optimum plans are produced, many spot decisions will be required while executing the plans. The on-line simulation offers a chance for some practice decision making.

On the level where it is most applicable the off-line simulation of military situations is in widespread use. A number of private businesses as well as military facilities are engaged in producing military simulations of this type. The Navy has a number of on-line simulations in use for training purposes. Many of these are an analog type and mose are available only at training facilities. Others are in use strictly to provide targets to a particular weapons system. All of these on-line simulations combined still reach only a small percentage of those who could profitably use them.

Until recently there was a very good reason for this. There were no suitable computers reasonably available. Although this reason is disappearing, another one remains. The line officer has not demanded an extensive on-line simulation capability from his on board computers. To some degree this is because he is not aware of the digital computer's ability in this area.

No parallel exists for the off-line simulation. The scientist is an active customer for the analytical simulation. He is well aware of the capabilities of the computer, and in many cases he is quite capable of producing his own simulations. The scientist or operations analyst has little personal need for a training device. It is not surprising then that off-line simulations are in wide use, are understood and

exploited rather fully. On-line simulations are relatively uncultivated.

The on-line simulation has some obvious disadvantages. It can be an extravagant and inefficient use of computer time. Although the example problem in the appendix requires less than two minutes for compilation and assembly, it can easily consume a full hour of computer time for just five or six practice torpedo approaches. Additionally, it shares almost all of the disadvantages of simulations in general, including the very important inability to fully verify the model with the real world.

5. Conclusions.

Writing these simulations and examining the results has made some observations possible. Certainly it raised some doubts about the future of simulation languages. If the quality of the output data is the chief concern, and it should be, we must observe that a change to a simulation language offers no advantage. An imaginative programmer with a good, present day, general purpose language can make a computer step through almost any combination of its basic instructions. A simulation language can do no more. The simulation language seeks to group and package those sequences of instructions most frequently used in simulation-like problems. The goal is to make it easier and faster to write simulation programs. For the same reason general purpose languages have packaged sequences of instructions useful for general problems. At the present time, these parallel efforts have covered quite a lot of common ground.

This study has found the only areas of unique coverage by the

simulation languages to include event processing, list processing, the existence of object type variables, and an ability to work with class membership. It was demonstrated that the latter two can be processed by a general purpose language with little more than a change in terminology. Event processing and list processing are worthy of more serious concern. List processing is by no means confined to simulations. In the future it may well be widely available in general purpose languages. At present, the general purpose language user might have to resort to a significant amount of extra coding to realize adequate handling of his events and lists.

A facility contemplating the adoption of a simulation language capability should ask some questions: What properties of the situations to be simulated can not be handled by a general purpose compiler? Does a simulation language meet these needs? Simulating the real world on a digital computer is difficult at best, but simulation languages do not remove all the problems. The improvements offered are in identifiable areas. Specific problems may benefit from only a few of these areas.

This suggests the creation of some suitable subroutines. The time spent by programmers in learning a new language might be more profitably spent in writing some versatile event sequencing and list processing subroutines. Properly tailored to the needs of a particular class of problems, these should offer significantly more efficient use of computer time. This and a continuing improvement of general purpose languages would leave little reason to acquire a special purpose simulation language.

The reader may feel that this is in reality an attack on the specific simulation language tested, MILITRAN. It is not meant to be. MILITRAN is an attractive language to use in many ways. It is as easy to learn to use as any general purpose language. The manuals are equal to any and superior to most in completeness, readability, and wealth of examples. Programming in the language is pleasant due to the flexibility and power that are simultaneously available. In fact MILITRAN has demonstrated some worthwhile improvements possible with general purpose languages. It is easier and more pleasant to program many general purpose problems in MILITRAN than it is in FORTRAN 63. No, the fault does not lie with MILITRAN.

The general purpose languages are not bad enough. They are very nearly adequate and in time they should become more so. Some compelling reason must be present to inspire an individual to learn a special purpose language. Additionally, he must have faith that the new language will have a long enough life to justify his investment of time. Will it be updated for future generations of computers? Is it presently available on virtually all of the present generation of computers? Any decision to change must be made in the face of these questions. This study has indicated that, for many users, the general purpose languages just do not possess serious enough shortcomings to compel a change to a special purpose language. The reason seems basic. Simulations of the type studied have not been found to be so significantly different from general problems to require a special language.

A reasonably realistic on-line simulation was demonstrated. It

required neither excessive hardware nor extensive programming effort. An attempt to compare on-line simulations and off-line simulations indicated that they were not necessarily competitors. The on-line simulation can complement the off-line simulation. The military line officer might well use the results of an off-line simulation to help him develop the best possible plans. He might also use the on-line simulation to gain practice in decision making in the face of some unpredictable combinations of developing situations that could be met while implementing the plans.

The Armed Forces have never been unaware of the value of on-line type simulations. Training facilities throughout the services have many excellent and heavily used simulators.

The operations analyst is certainly aware of the possibilities of off-line simulations. He exploits them quite fully whenever they can serve his needs.

An important disadvantage of the on-line simulation is removed when it is considered for use on a Navy ship. Here the inefficient use of computer time is of no concern. The computer, like everything and everyone aboard a warship, is only completely utilized in battle. In some applications the computer is so vital that redundancy must exist in the installed equipment. Certainly then at times, hopefully often, computer time is available for simulation purposes. Taking advantage of this time is a challenge. It may be well worth the expense and effort.

The modern warship with its complex equipment has complex tasks. As a part of a general purpose force it will have several different

complex tasks. The guided missile destroyer still has an ASW capability, yet it might well be required to spend long periods of time helping to maintain a naval blockade. If, during the blockade, this ship's captain feels the need to tone up his ASW skills, he has little recourse. An hour of practice decision making for him, at any time in the near future, is impossible. Arranging for another ship's services in a suitable area and also finding someone to fill his blockade station offers sufficient problems to ensure that his skills will continue to grow stale. The simulators at the training facilities are equally unavailable. Probably when training time is available the missile firing mission will receive priority. Clearly, a need exists.

The above situation is only slightly hypothetical. The patrol of a Polaris submarine offers no break during which the captain can practice fighting his ship against an attacker. A ships OOD has little chance, without hazarding his ship, to test his own ability to provide the correct orders to the helm in a complex tactical situation and then watch the result of those orders. Every line officer can add to this list.

The Navy works very hard to try to keep all of its wartime capabilities sharp. Cold war duties, a lack of time, the inability to obtain services, budget problems and too many other difficulties all combine to hold us short of our goal. These problems will never vanish. Neither will our tasks decrease in number nor grow less complex. Perhaps the on-line simulation can help us maintain a high state of readiness.

All an on-line simulation can do is to present a possible combination of situations, allow a person to make a decision in the face of these circumstances, and then generate the results of this decision. Properly used this is quite enough.

The on-line computer simulation won't replace anything. It fills a gap where nothing exists. We will still need all the training exercises we can get; but hopefully, even as operational commitments rise, we will not slip to a level of dire need between the exercises. Possibly the most elementary exercises can be simulated on a computer well enough so that available time at sea can be spent on the more beneficial advanced exercises. At least they can make it possible for every key individual to test his own readiness at any level, independently of the availability of target services, operating areas, breaks in own ships commitments, schools, or even other members of his own team.

The problems are large. These areas deserve and need further study: How much complexity and expense would be required to give present and proposed computer installations aboard the various ships a hardware capability for worthwhile simulations? How much complexity and expense could be saved if such a capability was planned in at the design stage? Is it within the scope of a USNPGS thesis to find the answer to the first question by actually designing a simulation feature into an afloat system? If the simulation ability did exist in a fleet unit would it not be well within the capability of USNPGS students to conceive and program high quality simulations for the system?

The problems are indeed large. So are the possibilities.

BIBLIOGRAPHY

1. Bagley, P. R. Improving problem-oriented language by stratifying it. The Computer Journal (British), v. 4, no. 3, October 1961: 217-221.

2. Barton, R. S. A critical review of the state of the programming art. Proceedings of the Spring Joint Computer Conference 1963: 169-177.

3. Buxton, J.N. and Laski, J. G. Control and simulation language. The Computer Journal (British), v. 5, October 1962: 194-199.

4. Computer Usage Company, Inc. Study of CDC 1604 programming systems. Final report. v. 1, February 1962.

5. Cotton, M. L. Tactical simulation methods. Proceedings of the Eighth Navy Sciences Symposium, May 1964: 76-85.

6. Gaskill, R. A. A versatile problem oriented language for engineers. IEEE Transactions on electronic computers, v. EC-13, August 1964: 415-421.

7. Huskey, H. D. A language for simulation. Symposium on simulation models: Methodology and applications to the behavioral sciences. South-Western Publishing Company, 1963: 13-25.

8. International Business Machines Corporation, Reference manual General Purpose System Simulator II. Anon. 1963.

9. Knuth, D. E. and McNeley, J. L. SOL--A symbolic language for general purpose system simulation. IEEE Transactions on Electronic Computers v. EC-13, August 1964: 401-415.

10. Krause, L. Simulation languages: evaluation and prescription. Paper presented at a seminar in decision theory at the Graduate School of Business, Stanford University, Palo Alto, California, June 1964.

11. Leach, G. H. and Perrella, A. J. A satellite computer system for the on-line analysis, control, and display. USNPGS thesis. June 1964.

12. Orchard-Hays, W. The evolution of programming systems. Proceedings of the IRE, January 1961, 283-295.

13. Orchard-Hays, W. The general problem of computing languages: their nature, function, and translation. Preprints of papers presented at the 16th national meeting of the ACM. September 1961: 2b-4(1) to 2b-4(4).

14. Pugh, A. L., III. DYNAMO users manual, second edition. The M.I.T. Press, May 1963.

15. Markowitz, B. H. and Karr, W. H. SIMSCRIPT a simulation programming language. The Rand Corporation, April 1963.

16. Simulations Council Inc. The art of simulation management. Anon. Simulation: The journal of Simulations Council Inc., v. 3, October 1963: 3-15.

17. Systems Development Corporation. Digital simulation techniques applied to a communications system by R. Archambault et. al. Technical Memorandum TM(L)-632, July 1961.

18. Systems Development Corporation. Simulation with digital computers, by P. Peach. Report SP-1390, October 1963.

19. Systems Development Corporation. The use of simulation in management analysis--a survey and bibliography, by D. G. Malcolm. Report SP-88, November 1958.

20. Systems Research Group, Inc. MILITRAN operations manual, IBM 7090/94. 1964.

21. Systems Research Group, Inc. MILITRAN programming manual. 1964.

22. Systems Research Group, Inc. MILITRAN reference manual. 1964.

APPENDIX I

MILITRAN LANGUAGE DESCRIPTION

Chapter III of Reference 4 is a description, in outline form, of four computer languages; COBOL, FORTRAN, JOVIAL, and NELIAC. All of the languages are described according to a similar format in order that feature by feature comparisons among the language can be made.

Although MILITRAN is currently available only on IBM 7090/94 computers, serious consideration has been given to producing a version for the CDC 1604. In order to facilitate a comparison with the above languages MILITRAN is herein described in this same format.

A brief description of a pliant and complex language such as MILITRAN implies an incomplete description. The essentials of MILITRAN are here. There are four instructions that are not explained (ATTACKER, INDEX, TARGET, and TIME). Understanding these requires a more detailed explanation of MILITRAN than the scope of this paper allows. These four instructions add convenience, but not substance to the language. Their employment and the very many variations and subtleties of the other instructions may be discovered by those interested enough to read references 20, 21, and 22.

MILITRAN

LANGUAGE DESCRIPTION

I. ALPHABET

A. Letters — A thru Z

B. Numerals — 0 thru 9

C. Special — period, comma, (,), =, +, -, *, /

II. INFORMATION

A. Numeric
   - Octal
   - Real (floating point)
   - Integer (fixed point)

B. Alphabetic — Letters only

C. Alphanumeric — Letters or numbers

D. Special Characters — The remainder of the alphabet stated above.

III. ARITHMETIC OPERATORS

+; -; *; /; .P.; =

IV. RELATIONAL OPERATORS

.E.; .NE.; .G.; .GE.; .L.; .LE.

V. LOGICAL OPERATORS

.NOT.; .AND.; .OR.; .EXOR.; .EQV.

VI. SEQUENTIAL OPERATORS

A. GO TO (Unconditional) Statement

An unconditional branch to another part of the program.

e.g.; GO TO JOB 104

(JOB 104 is a statement label appearing somewhere in the program)

B. IF Statement

For conditional branching to other parts of the program using a logical expression as the criterion for branching.

e.g., IF (CONTACT .IN. ENEMY), ATTACK 1, SEARCH 14

(If the statement is true, the program object CONTACT is a member of the Class ENEMY control will be transferred to the statement labeled ATTACK 1. If the expression is false control will be transferred to SEARCH 14. If the second label is omitted, control will pass to the next statement.)

e.g., IF (RANGE .GE. 50000), RESEARCH

(If the logical expression is true control passes to the statement labeled RESEARCH. Otherwise, the next statement will be executed.

C. UNLESS Statement

This is similar to an IF statement, but the order of statement labels is reversed. Thus, a true logical expression and an omitted second label will cause the next statement to be executed.

e.g., UNLESS (AMMO .LE. 100), FIRE

(The logical expression will be evatuated and if false the statement labeled FIRE will be executed. If the expression is true the next instruction will be executed.

D. DO Statement (Indexed)

This repeats a sequence of statements until a logical expression is true and advances an index by any specified increment for each separate iteration of the sequence of statements in the loop.

e.g., DO (STEP 10) UNTIL (TARGET (I) .IN. ATTACKABLE) .OR. (I. G. 100), I = 1, 100

(If none of the 100 TARGETS are in the ATTACKABLE class the loops will be terminated when I reaches 100.)

E. DO Statement (Not Indexed)

The second form of the DO statement operates only on object elements.

e.g., DO (B 10) FOR SHIP .IN. FORMATION

(SHIP is a variable. It is first set to equal the first member of a class called FORMATION. Instructions within the range of the DO loop are then executed. When label B 10 is reached SHIP is set to equal the next member of FORMATION and the loop is repeated. The iteration terminates when SHIP has taken on the identify of each member of FORMATION.)

F. CONTINUE Statement

A dummy statement used to provide a statement number at the proper place in a statement.

G. STOP

Immediately terminates the object program. No restart is possible.

H. PAUSE j

Temporarily terminates the object program. Pressing the start key will continue the object program at the next instruction. "j" can be blank or an octal integer from 1 to 5 digits. If used "j" will appear on the console when the computer stops.

VII. FILE OPERATORS

A. READ Statement

e.g., READ (4, A100) A, B, C, D

This causes data to be read in from input unit #4 in the format appearing at label A100. The list of data to be read is A, B, C, D.

B. WRITE Statement

e.g., WRITE (5, A210) TARGETS

This is similar to the READ statement, but the data is written on the designated output unit.

C. READWRITE Statement

e.g., READWRITE (4, A100, 5, A101) A, B, C, D

This is identical to the execution of the following two instructions

READ (4, A100) A, B, C, D

WRITE (5, A101) A, B, C, D

in the order shown.

D. BINARY READ Statement

e.g., BINARY READ (4) VELOCITY

The binary information on tape unit #4 is read to the location specified by the list VELOCITY.

E. BINARY WRITE Statement

e.g., BINARY WRITE (5) VELOCITY

The object program writes the binary information contained in the location specified by VELOCITY onto the output tape #5.

F. END FILE RETURN Statement

e. g., END FILE RETURN (A101)

When an end of file mark is encountered control is shifted to the statement with label A101.

G. END RECORD RETURN Statement

e.g., END RECORD RETURN (A101)

This is used to shift control when the end of a logical record may be encountered before a BINARY READ has been completed. Again A101 is a statement label and control goes to this statement.

H. BACKSPACE Statement

e.g., BACKSPACE (5)

If a binary tape is on tape unit #5 it is moved backward one logical record. A BCD tape on unit #5 would be moved back one physical record.

I. BACKSPACE FILE Statement

e.g., BACKSPACE FILE (5)

The tape on unit #5 is moved back until an end of file or load point is encountered.

J. END FILE Statement

e.g., END FILE (OUT)

An end file mark is written on the tape mounted on the tape unit designated OUT by the object program.

K. REWIND Statement

e.g., REWIND (4)

The tape mounted on tape unit #4 is rewound to the load point.

L. UNLOAD Statement

e.g., UNLOAD (5)

The object program rewinds tape unit #5 and puts it in the automatic unload status.

VIII. FUNCTIONAL MODIFIERS

A. EACH

This, used in a CLASS declaration (see XI G.) specifies that members of a set of object elements are not identical and for this reason cannot be represented by a single member of the group.

e.g., CLASS (DANGEROUS) CONTAINS EACH * UBOAT, EACH * QSHIP

B. LST

This modifier is employed with list processing statements. Its action is explained in item XII. N.

C. GST

This modifier, similar to the previous one, is explained in item XII. O.

IX. MACHINE LANGUAGE STATEMENTS

A MILITRAN program is first translated to a FAP program. (At present, no other versons are available.) At this point the coding may be altered with machine language statements. FAP coded subroutines, conforming to the MILITRAN calling sequence, may also be employed.

X. GENERAL STATEMENTS

A. PROCEDURE Statement

This statement is the entry point to a procedure and defines that which follows to be a procedure.

e.g., PROCEDURE MAPPING (G, L)

The name of the procedure is "MAPPING" and each of the letters is a dummy name for the arguments to be supplied when the procedure is called.

B. RETURN Statement

At every point where a procedure can terminate there must be a RETURN statement.

e.g., RETURN RANGE

If the RETURN statement is followed by an expression, the value of that expression is placed in the accumulator and control is returned to the main program. If RETURN is used alone, the contents of the accumulator is unspecified.

C. EXECUTE Statement

Control is transferred to a named subroutine by this statement.

e.g., EXECUTE MAPPING (RANGE BEARING)

Here control will be shifted to subroutine MAPPING with the current value of the variables RANGE and BEARING as actual arguments.

XI. DECLARATIONS (Data Description)

A. COMMON Statement

This assigns variables, arrays, vectors, and lists to the common

storage field.

B. REAL Statement

This assigns both the mode and the dimension of an array.

e.g., REAL ENEMY (PLANES, 2)

The real array ENEMY is declared. One dimension is a "symbolic" dimension that need only be specified at object time. In this example the other dimension will always be 2.

C. INTEGER Statement

The use of this statement is identical to that of the REAL statement, but declares INTEGER arrays.

D. LOGICAL Statement

This, like the above two statements, declares the mode and dimension of a LOGICAL array.

E. OBJECT Statement

The OBJECT declaration generates a nonnumeric element which is the name of an identifier.

e.g., OBJECT UBOAT (10)

An element named UBOAT has been generated and also UBOAT consists of a set of 10 ships.

F. PROGRAM OBJECT Statement

A declaration of this type allows a variable to exist which can take on the value of names.

e.g., PROGRAM OBJECT ENEMY CRAFT

This specifies a variable which can take on the name of any object in the program.

G. CLASS Statement

This statement declares that certain properties of designated objects or the objects themselves have common characteristics. These characteristics form a set.

e.g., CLASS (DANGEROUS) CONTAINS UBOAT, QSHIP

Thus the objects UBOAT and QSHIP have been declared to belong to a CLASS called DANGEROUS.

H. NORMAL MODE Statement

This declaration allows a programmer to give all variables beginning with a single letter the same mode.

e.g., NORMAL MODE INTEGER (I, J), LOGICAL (L)

With no further declaration statements all variables beginning with I or J are INTEGER and all of the variables starting with L are LOGICAL mode.

I. VECTOR Statement

A group of arrays may comprise a vector.

e.g., VECTOR Q((QR, QS, QT), 4, 5)

The vector Q has three components; QR, QS, and QT. Each of these is a two dimensional array with 20 storage spaces reserved.

J. LIST Declaration

See item XII.

K. FORMAT Statements

For describing input-output format and code conversion.

(Note: b denotes blanks.)

1. I - Control

   Integer decimal external/Fixed point

   e.g., (I5) ⟶ 1 2 3 4 5

2. E - Control

   Floating decimal external/Floating point. (with exponent)

   e.g., (E 10.3) ⟶ b0. 1 2 3 E b 0 4

3. F - Control

   Floating decimal external/Floating point. (without exponent)

   e.g., (F 7.2) ⟶ 1 2 3 4. 0 0

4. O - Control

   Octal digits external/Octal

   e.g., (O8) ⟶ 1 2 3 4 5 6 7 0

5. J - Control

   Object elements

   e.g., (J 9. 2) ⟶ UBOAT(10)

6. L - Control

   Logical values

   e.g., (L2) ⇒ bT

7. A - Control

   Alphanumeric external/6 - bit code

   e.g., (A6, A1) ⟶ TAKEOFF

8. H - Control

   For alphanumeric headings

   e.g., 15HPAGE NUMBER ONE

9. X - Control

   For external spacing

   e.g., (I2, 3X, I3) ⟶ 1 2 b b b 3 4 5

X. OBJECT RELATIONAL OPERATORS

Object relational operators may operate only on object elements.

A. IN Operator

   e.g., UBOAT (7) .IN. WOLFPACK

   The above statement is either true or false and is evaluated as such by the program. In general the position held by UBOAT (7) may be filled by a PROGRAM OBJECT, an OBJECT, a member of a CLASS, or the name of a CLASS. The place held by WOLFPACK must be filled by an OBJECT or the name of a CLASS.

B. IS Operator

   e.g., UBOAT (7) .IS. TARGET

   This statement is also either true or false. It can be true only if the object elements are identical. These elements may be OBJECTS, PROGRAM OBJECTS, or members of a CLASS.

XI. EVENT PROCESSING STATEMENTS

A running MILITRAN program usually proceeds through loops of EVENTS. Those EVENTS which make up the active loop change constantly as the program progresses.

Event processing statements dictate which EVENTS will be within the loop and thereby control the way in which the program proceeds.

A. PERMANENT EVENT Statement

A PERMANENT EVENT is one which occurs at fixed intervals of time without regard to the eventualities of the problem. Thus, each complete loop of the problem will include every PERMANENT EVENT.

B. CONTINGENT EVENT Statement

CONTINGENT EVENTs contain lines of programming which are to be executed only when certain prescribed conditions exist. When these conditions do exist, a flag is set to cause the appropriate CONTINGENT EVENT to be placed in the active loop for a futute execution.

C. NEXT EVENT Statement

This statement indicates an EVENT has been completed and the next appropriate event is to be executed. Many possible uses of this statement exist. A full explanation is beyond the scope of this paper.

D. NEXT EVENT EXCEPT Statement

This statement acts much as the NEXT EVENT statement; but, as its name implies, it allows the programmer to specifically exclude possible courses of action at a particular juncture.

E. END Statement

The END statement defines the end of processing for a specific event. If it can be reached by program logic it is interpreted as an unmodified NEXT EVENT.

F. END CONTINGENT EVENTS Statement

This special type of statement, employed early in a MILITRAN program and prior to starting any of the events within the loops, specifies where the program is to transfer control when the conditions for completing the last loop have occurred.

e.g., END CONTINGENT EVENTS (LAST)

LAST is a statement label. Control will be transferred there when

all desired loops have been completed.

G. EPSILON Function

This, one of several open functions, facilitates the flagging of CONTINGENT EVENTS for execution. It replaces a value with a new value larger by the smallest possible increment within the computer.

XII. LIST PROCESSING STATEMENTS

A. LIST Statement

This only declares the dimension of a LIST.

e.g., LIST SHIPS ((DESTROYER, CRUISER, CARRIER), 10)

A LIST named SHIPS is defined. Each of the components has a dimension of 10 - thus the LIST SHIPS has a total of 30 locations reserved.

B. LENGTH Function

e.g., LENGTH (SHIPS)

The function returns the current number of entries in the LIST.

C. RESET LENGTH Statement

The length of any LIST may be unconditionally set to any desired length by this statement.

D. PLACE Statement

Any component may be entered in a LIST by using this statement.

E. REMOVE ENTRY Statement

This statement allows the program to remove any entry from a LIST.

F. PLACE ENTRY Statement

An entry in one LIST may be entered in another LIST by this statement.

G. REPLACE ENTRY Statement

One entry on a LIST may be deleted and a specified entry inserted in its place with the REPLACE ENTRY statement.

H. REPLACE ENTRY BY ENTRY Statement

An existing entry in one LIST may be used to replace another

entry in the same or a different LIST when this statement is employed.

I. REMOVE FROM Statement

This statement, used with a logical expression, allows a program to remove any or all entries on a specified LIST which cause the logical statement to be evaluated as true.

J. REPLACE BY Statement

This statement must also be accompanied by a logical expression. When an entry causes the logical expression to be evaluated as true, the entry on the specified LIST is replaced by a stated entry.

K. REPLACE BY ENTRY Statement

This is similar to the previous statement; but, instead of stating the new entry, allows another entry in some LIST to be used as the new entry.

L. MINIMUM INDEX Statement

A logical expression is evaluated with every entry in a specified LIST. The lowest index number in the LIST which satisfies the expression is returned.

M. RANDOM INDEX Statement

This is similar to the above entry; but, as the name implies, a random index number among those satisfying the logical expression is returned.

N. LST Statement

If this is employed with statements I through K above, it imposes additional controls on the statement. It may be used preceding only one of the logical expressions within the statement. It then allows modification only of the LIST entry whose corresponding component is least among those satisfying the logical expression.

C. GST Statement

This is similar to the above statement but allows operation only on the LIST entry with the corresponding largest element.

APPENDIX II

OFF-LINE SIMULATION PROGRAMS

This appendix contains the flow charts and program listings for two very similar simulations. The common problem being simulated concerns a barrier submarine at a fixed station of grid position X = 5., Y = 6. This is the center of a lane as sketched below.

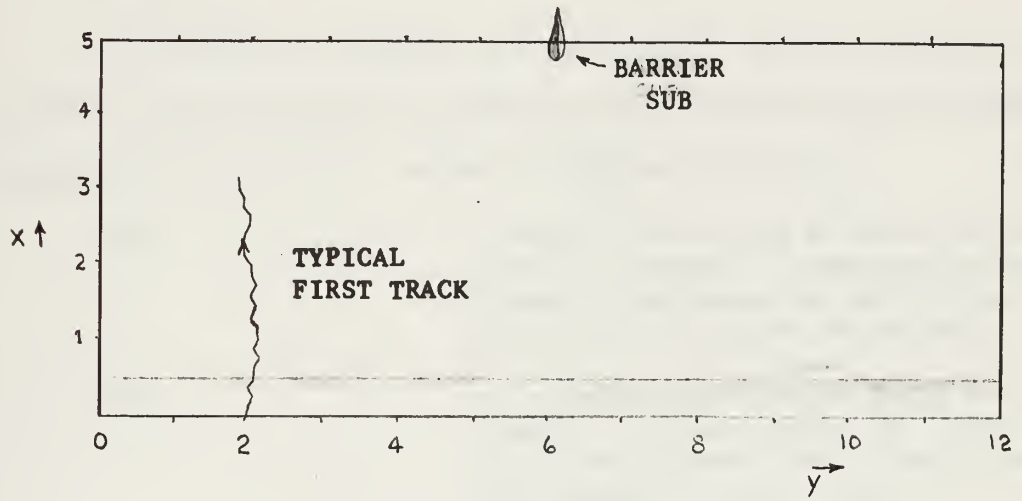


FIGURE 1

Off-line simulation playing area

A series of transitors of random type (merchant, submarine, destroyer, nuclear submarine, or fishing boat) commence a step by step transit from the line X = 0 toward X = 5. The first transitor commences his run from Y = 2. After the first transitor completes his run successive runs are initiated from increasingly larger values of Y position. The simulation terminates when a transit starts from a Y exceeding twelve.

The flow charts and program comments explain the rest of the simulation. The MILITRAN program names should be self-explanatory

with only the exceptions noted below:

SON TAB	A table of probabilities of detection for each grid square for each of four different sonar conditions.
CUR SON	The part of the above SON TAB which is applicable for the current sonar condition.
SON STAT SONTSTAT SONCSTAT SONAR COND	Various dummy arguments all equal to the current sonar condition (an integer between 1 and 4).

The FORTRAN 63 program simulates exactly the same situation. The variable names and names of arrays parallel those of the MILITRAN program where possible. Notable exception include:

LHIST	A logical array with an entry for each transit. Column 1 is whether or not the transitor was attacked, and column 2 is whether or not he was destroyed.
IHIST	An integer array with an entry for each transit. Column 1 gives the sonar condition which prevailed, column 2 is total times detected, and column 3 is the type of transitor.
ITRSUC	An array consisting of the numbers of the successful transits.
LCLASS	A logical array to record class membership. See Figure 2 below.
TRANSC	An array giving the assumed transitor characteristics. See Figure 3 below.
RN	A random number.

TYPE SHIP	CLASS ALL	CLASS ENEMY	CLASS DANGEROUS	CLASS SUBMARINE
SUB	1	1	1	1
NUC	1	1	1	1
FSH	1	1	0	0
DDE	1	1	1	0
MER	1	0	0	0

FIGURE 2

Array LCLASS

TYPE SHIP	KIND	SPEED RATIO	KILL FACTOR	DETECT FACTOR
SUB	1.0	1.0	1.0	.9
NUC	2.0	3.0	.8	.9
FSH	3.0	.5	1.0	1.0
DDE	4.0	4.0	1.0	1.0
MER	5.0	4.0	1.0	1.0

FIGURE 3

Array TRANSC

Both simulations output the same data in the same format. This consists of two parts. Each individual event during the simulation is reported, and a summary of the results of each transit is maintained. Figure 4 is a sample of a few lines of each type of output.

HISTORY OF EVENTS

EVENT NUMBER	TRANSIT NUMBER	X	Y	TYPE SHIP	DESCRIPTION OF EVENT
71	44	5.24	11.44	MER	SAFE TRANSIT, NO DAMAGE ONLY 14 DETECTIONS
72	45	4.18	10.89	NUC	TRANSITOR SUNK
73	46	4.10	11.18	SUB	ATTACK ABORTED DUE TO SSK MATL FAILURE NO CHANCE TO REATTACK
74	46	5.00	11.18	SUB	SAFE TRANSIT, NO DAMAGE ONLY 32 DETECTIONS

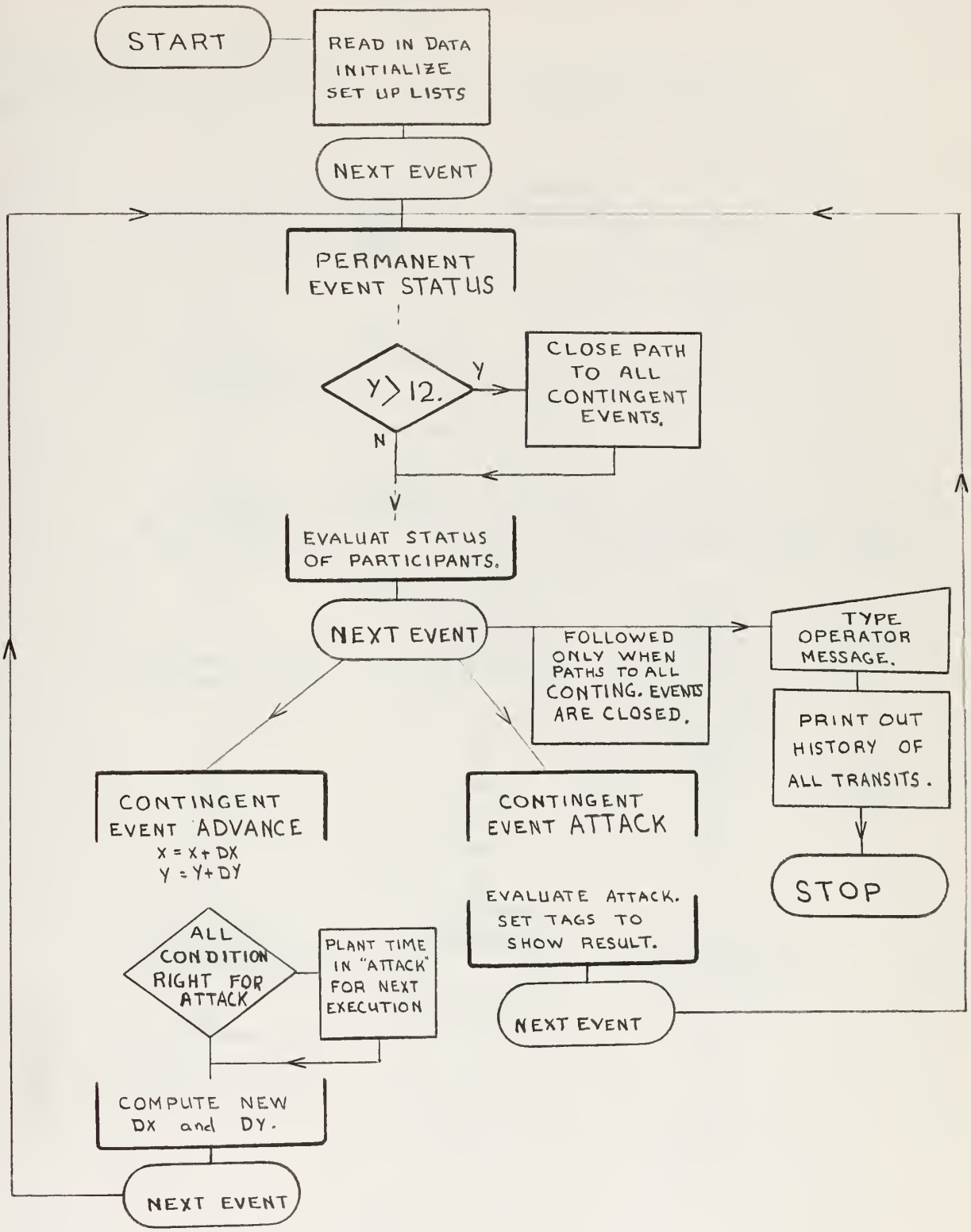
HISTORY OF ALL TRANSITS

TRANSIT NUMBER	SONAR CONDITION	TIMES DETECTED	ATTACK	DESTROYED	TYPE
1	1	1	0	0	MER
2	1	4	0	0	DDE
3	1	18	1	1	SUB
4	3	64	1	1	FSH
5	4	65	1	1	FSH
6	4	12	0	0	MER
7	3	24	1	1	SUB
8	3	63	1	1	FSH
9	2	11	1	1	NUC
10	2	9	1	1	DDE

FIGURE 4

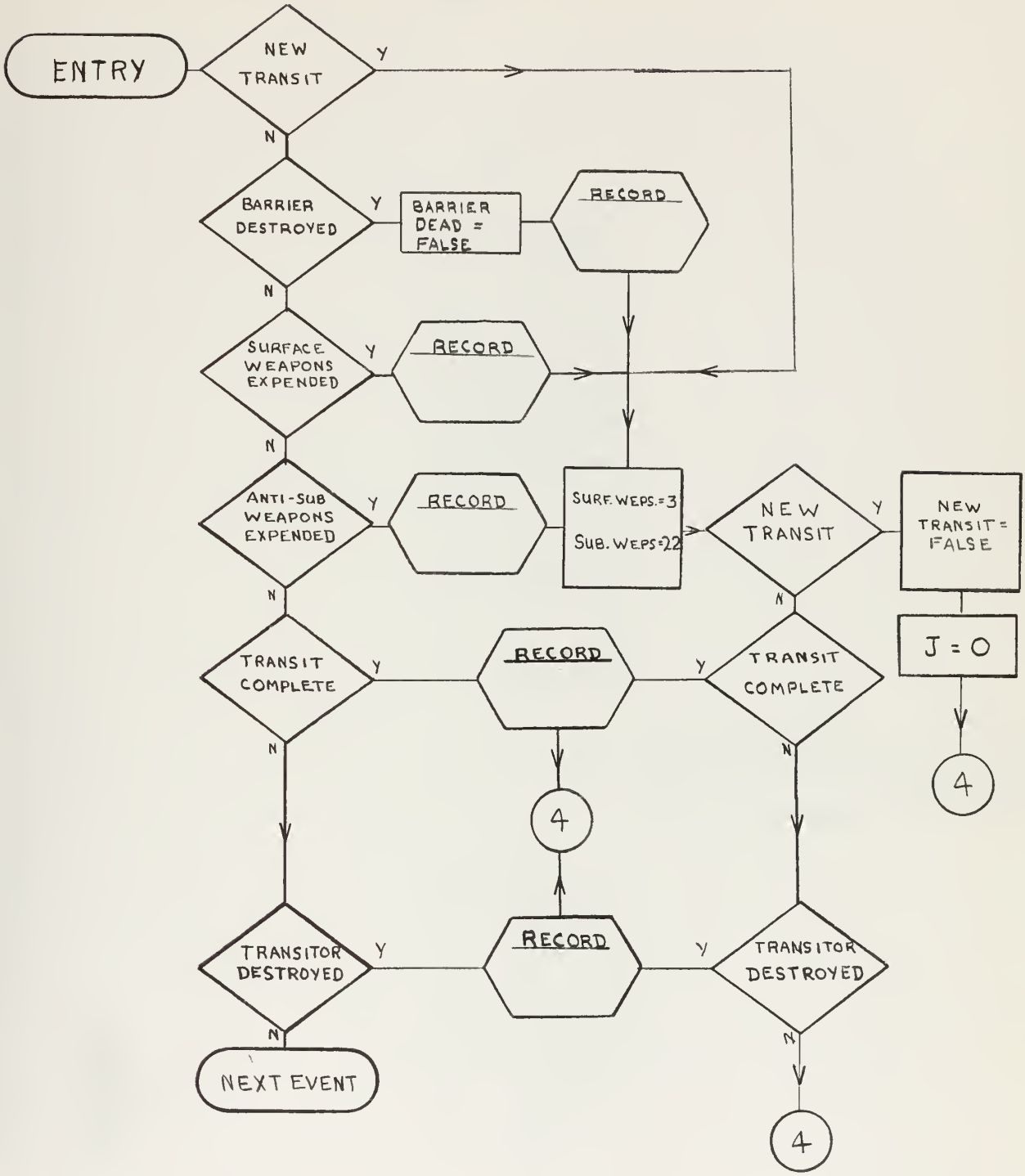
Sample of output of off-line simulation

MILITRAN (SKELETON)



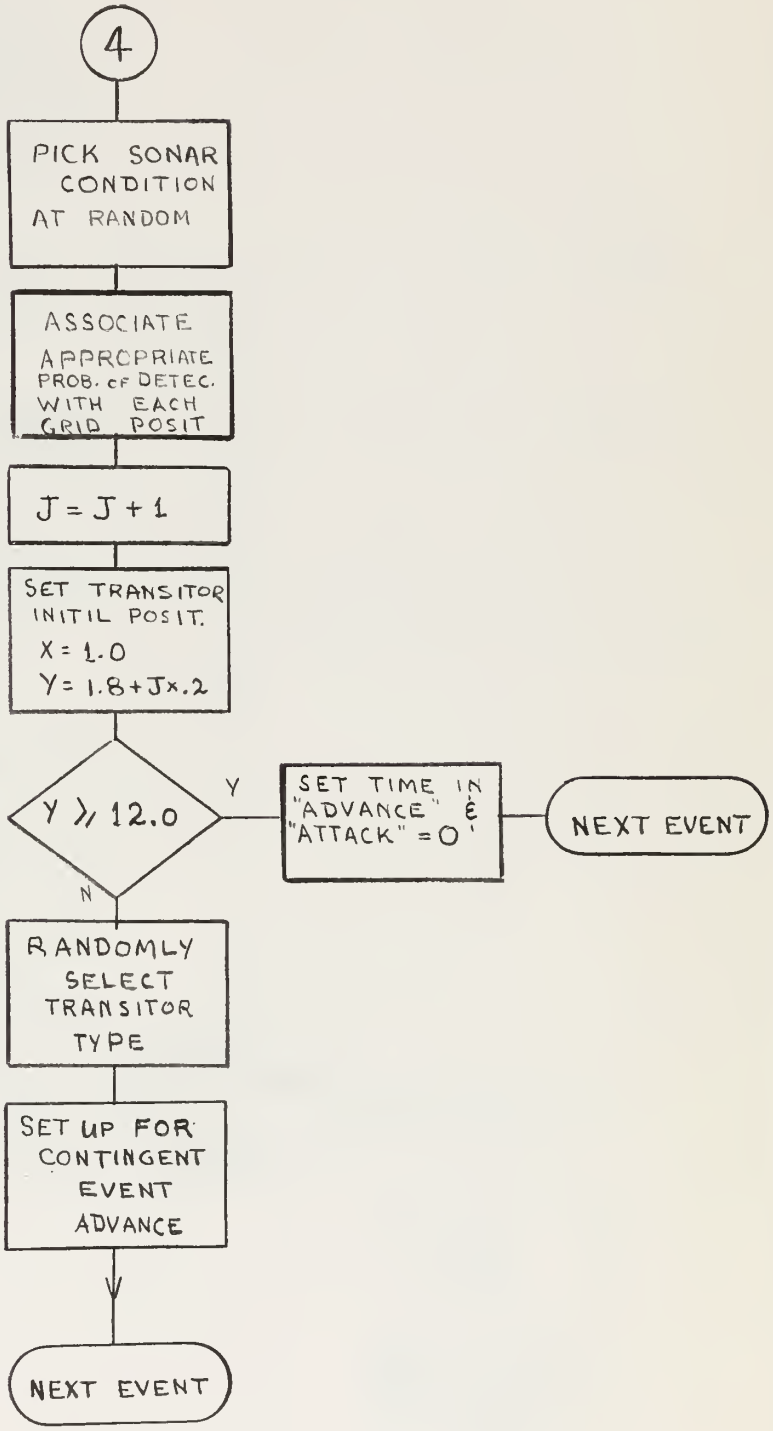
MILITRAN PERMANENT EVENT STATUS

CHART 1

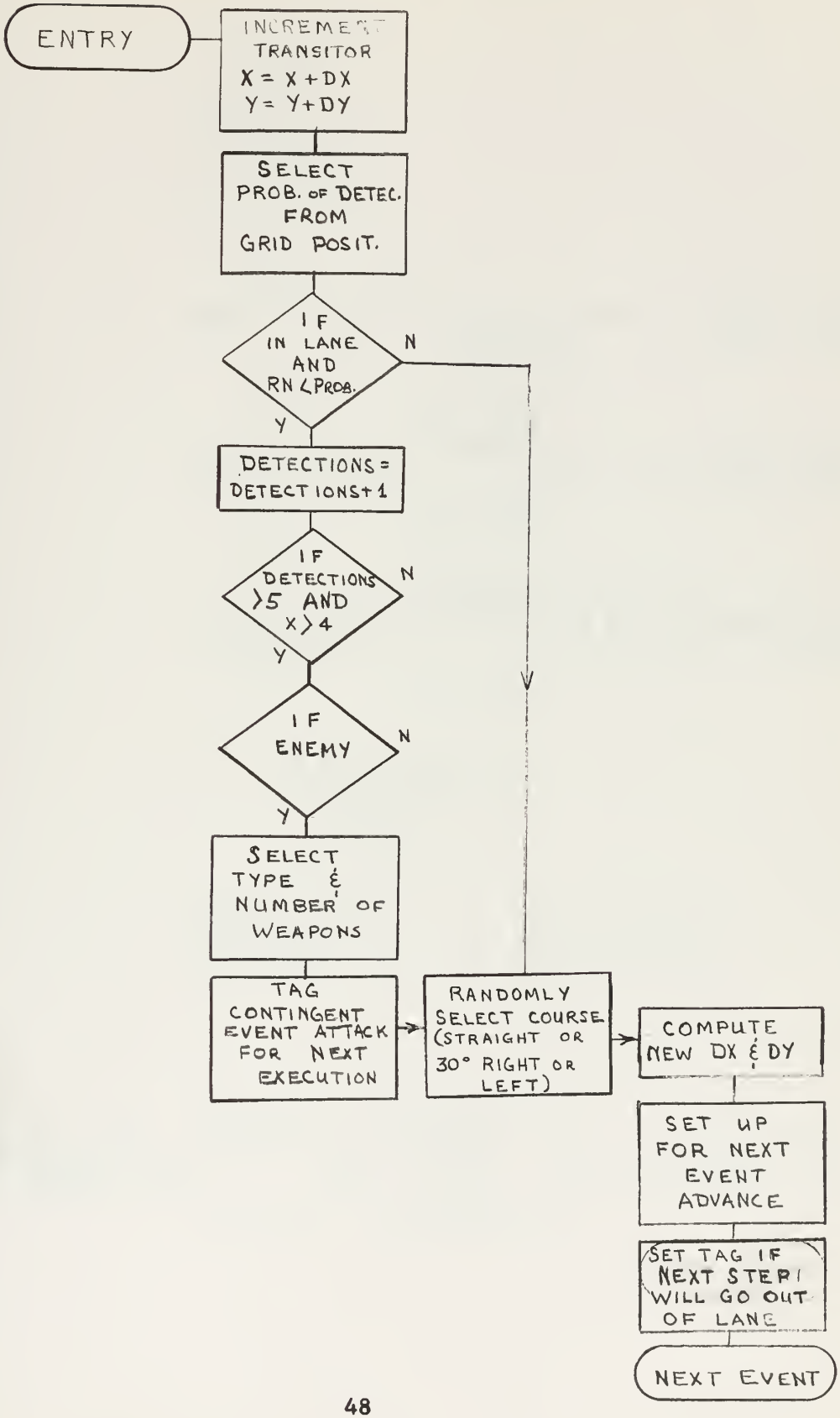


MILITRAN PERMANENT EVENT STATUS

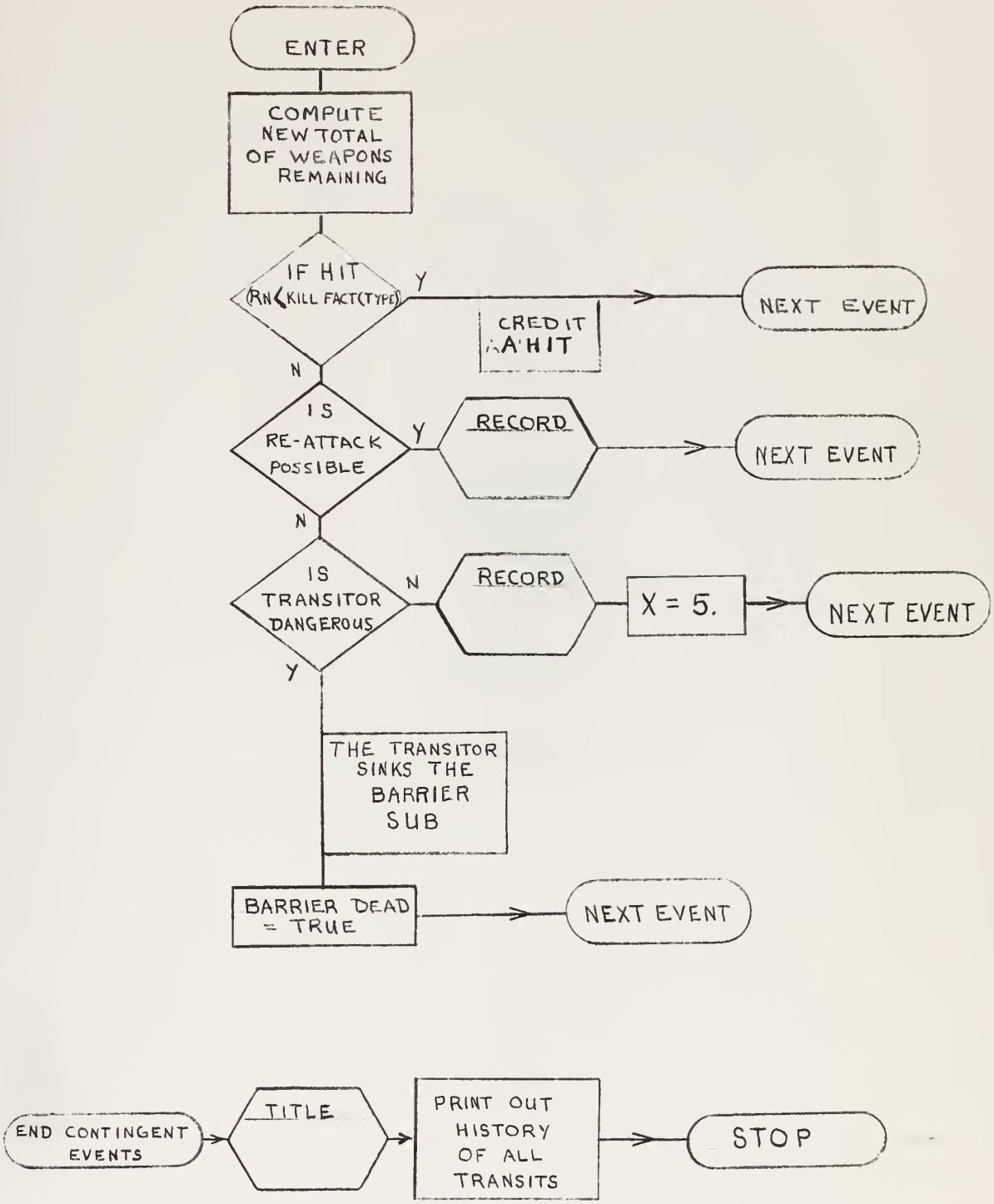
CHART 2


4
PICK SONAR CONDITION AT RANDOM
ASSOCIATE APPROPRIATE PROB. OF DETEC. WITH EACH GRID POSIT
J = J + 1
SET TRANSITOR INITIL POSIT.
X = 1.0
Y = 1.8 + J x .2
Y ⩾ 12.0
Y
SET TIME IN "ADVANCE" & "ATTACK" = 0
NEXT EVENT
N
RANDOMLY SELECT TRANSITOR TYPE
SET UP FOR CONTINGENT EVENT ADVANCE
NEXT EVENT

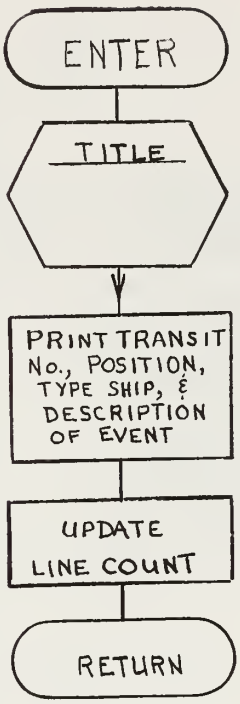
MILITRAN CONTINGENT EVENT ADVANCE



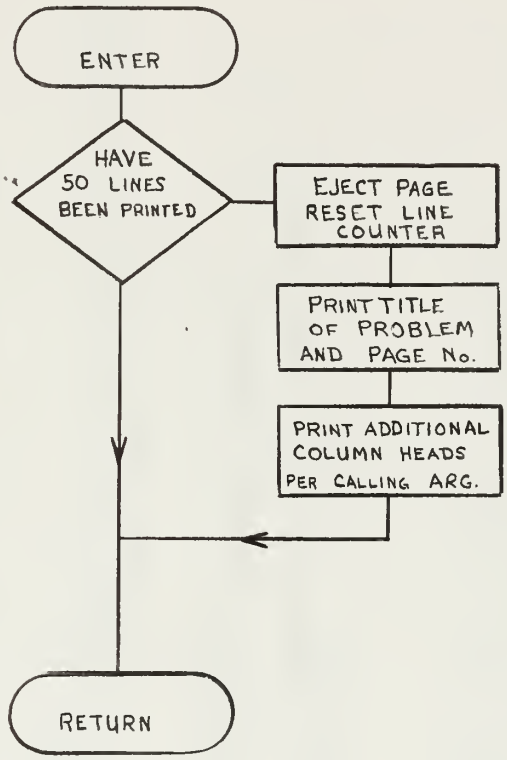
MILITRAN CONTINGENT EVENT ATTACK



MILITRAN PROCEDURE RECORD



MILITRAN PROCEDURE TITLE


ENTER
HAVE 50 LINES BEEN PRINTED
EJECT PAGE RESET LINE COUNTER
PRINT TITLE OF PROBLEM AND PAGE No.
PRINT ADDITIONAL COLUMN HEADS PER CALLING ARG.
RETURN

```
X  SINGLE SUBMARINE BARRIER
X MILITRAN PILOT SIMULATION

X BY JAMES E. JOHNSON  LT,USN

                                                                    SBM00010
X MAIN PROGRAM                                                      SBM00020
                                                                    SBM00030
                                                                    SBM00040
  COMMON ... BLOCK A ...                                            SBM00050
1PROBLEM NUMBER, PAGE, LINE, IN, OUT                                SBM00060
      INTEGER PROBLEM NUMBER, PAGE, LINE, IN, OUT                   SBM00070
                                                                    SBM00080
  COMMON   ...BLOCK B...                                            SBM00090
1EVENT NUMBER, TYPE, NEXT EVENT, J, X, Y, TRANS HIS, LLTHIS         SBM00100
      LIST TRANS HIS ((SONAR COND, TIMES DETECTED,ATTACKED,         SBM00110
1                     DESTROYED, NAME), LLTHIS)                     SBM00120
                                                                    SBM00130
      PROGRAM OBJECT NAME                                           SBM00140
      LOGICAL ATTACKED, DESTROYED, TRANS HIS                        SBM00150
                                                                    SBM00160
      INTEGER TYPE, TIMES DETECTED, EVENT NUMBER,J, K               SBM00170
      INTEGER SONAR COND                                            SBM00171
                                                                    SBM00180
                                                                    SBM00190
                                                                    SBM00200
  COMMON    ...BLOCK C...                                           SBM00210
1          SUBM,SURF, SURF WEAPON, SUB WEAPON,                      SBM00220
2          SON TAB, CUR SON, SUCCESSFUL TRANSIT,                    SBM00230
3          TRANSITOR,CNAME,SUB,NUC,MER,DDE,FSH,ALL,BRAND            SBM00240
4          NEW,SUBMARINE,ENEMY,DANGEROUS,LANE,BARRIER DEAD,         SBM00250
5          LLST                                                     SBM00260
                                                                    SBM00270
      LIST SURF WEAPON ((WA), 9),                                   SBM00280
1          SUB WEAPON  ((WB),22),                                   SBM00290
2          SON TAB ((SONTSTAT,A,B,C,D,E,F,G,H,O,P),16),             SBM00300
3          CUR SON ((SONCSTAT,CA,CB,CC,CD,CE,CF,CG,CH,CO,           SBM00310
```

```
     4                    CP),4)                                          SBM00320
                                                                          SBM00330
          LIST SUCCESSFUL TRANSIT ((TRANSIT NUMBER, SHIP,                 SBM00340
     1                    FINALX, FINALY), LLST),                        SBM00350
     2          TRANSITOR ((KIND, SPEED RATIO, KILL FACTOR,              SBM00360
     3                      DETECT FACTOR), ALL)                        SBM00370
                                                                          SBM00380
               INTEGER SON STAT,I, SONTSTAT                               SBM00390
     1            SONCSTAT, TRANSIT NUMBER                              SBM00400
                                                                          SBM00410
               PROGRAM OBJECT CNAME,SHIP , KIND, WEAPON TYPE              SBM00420
                                                                          SBM00430
               OBJECT SUB (1), NUC(1), FSH (1), DDE (1), MER (1)          SBM00440
     1            , SUBM (1), SURF (1)                                  SBM00450
                                                                          SBM00460
               LOGICAL LANE, BARRIER DEAD , BRAND NEW                     SBM00470
                                                                          SBM00480
                    CLASS (ALL) CONTAINS SUB, NUC, FSH, DDE, MER          SBM00481
                    CLASS(SUBMARINE) CONTAINS  SUB, NUC                   SBM00490
                    CLASS(ENEMY) CONTAINS SUB,NUC, FSH, DDE               SBM00500
                    CLASS(DANGEROUS) CONTAINS  SUB,NUC, DDE               SBM00510
                                                                          SBM00520
                                                                          SBM00530
          ...PREOPENING                                                   SBM00540
          PAGE=0                                                          SBM00550
          IN=5                                                            SBM00560
          OUT=6                                                           SBM00570
          READ (IN, FORM 0) PROBLEM NUMBER                                SBM00571
FORM 0    FORMAT (I5)                                                     SBM00572
          LINE=0                                                          SBM00580
          EVENT NUMBER =0                                                 SBM00590
          LANE=TRUE                                                       SBM00600
          BARRIER DEAD = FALSE                                            SBM00601
          BRAND NEW = TRUE      ...TAG TO INITIALIZE FIRST TRANSIT        SBM00610
          TIME= .001                                                      SBM00620
          PLACE (TIME, NUC(1), 0, 0) IN ADVANCE                           SBM00621
          RESET LENGTH (ATTACK) TO 1                                      SBM00622
```

```
                ATTACK TIME (1) =0                                                  SBM00623
                END CONTINGENT EVENTS(FINAL)                                        SBM00630
                ...BUILD TRANSITOR TABLE                                            SBM00640
                PLACE (SUB (1), 1.0, 1.0, 0.9) IN TRANSITOR                         SBM00650
                PLACE (NUC (1), 3.0, 0.8, 0.9) IN TRANSITOR                         SBM00660
                PLACE (FSH (1), 0.5, 1.0, 1.0) IN TRANSITOR                         SBM00670
                PLACE (DDE (1), 4.0, 1.0, 1.0) IN TRANSITOR                         SBM00680
                PLACE (MER (1), 4.0, 1.0, 1.0) IN TRANSITOR                         SBM00690
                ...BUILD SONAR TABLE LIST (SON TAB) AND TABULATE                    SBM00700
                EXECUTE TITLE (FORM 1B)                                             SBM00710
                LINE =0                                                             SBM00711
 FORM 1B        FORMAT (1H0  50X  21HSONAR TABLE (SON TAB) //)                      SBM00720
                DO (ABC 1) UNTIL I .G. 16,I                                         SBM00730
ABC 1           READWRITE (IN, FORM 2A, OUT, FORM 2B) SONTSTAT(I), A(I),            SBM00740
               1          B(I), C(I), D(I), E(I), F(I), G(I), H(I), O(I),          SBM00741
               2          P(I)                                                     SBM00750
 FORM 2A        FORMAT (I12, 10F6.2)                                                SBM00760
 FORM 2B        FORMAT (10H0SON STAT  I3, 7X 10F10.2)                               SBM00770
                RESET LENGTH (SON TAB) TO 16                                        SBM00780
                ...BUILD A LIST OF CURRENT SONAR CONDITIONS                         SBM00790
                DO (FIX 1) UNTIL I .G. 4, I                                         SBM00800
 FIX 1          PLACE ENTRY SON TAB (I) IN CUR SON                                  SBM00810
                NEXT EVENT          ...SHIFT CONTROL TO EVENTS                      SBM00820
 FINAL          WRITE (PRINTER, ON LINE)                                            SBM00830
 ON LINE        FORMAT (////23H ALL TRANSITS COMPLETE.)                             SBM00840
                LINE= 0                                                             SBM00850
   WRAP         EXECUTE TITLE (FORM 3A)                                             SBM00860
                WRITE (OUT, FORM 3B) ((I,((TRANS HIS(I,K))UNTIL K.G.5, K))          SBM00870
               1                          UNTIL I .G. LENGTH(TRANS HIS), I)SBM00871
   UP           LINE= 0                                                             SBM00880
                EXECUTE TITLE (FORM 3C)                                             SBM00890
DATA            WRITE (OUT, FORM 3D)(((SUCCESSFUL TRANSIT (I,K))                    SBM00900
               1                        UNTIL K.G.4, K)                             SBM00910
               2                    UNTIL I .G. LENGTH(SUCCESSFUL TRANSIT), I)SBM00911
                STOP          ...  THIS IS A NORMAL EXIT FOR COMPLETE PROGRAM SBM00920
 FORM 3A        FORMAT (1H0 48X 23HHISTORY OF ALL TRANSITS/1H015X7HTRANSIT  SBM00930
               1       6X  5HSONAR, 10X  5HTIMES,  8X  10HATTACKED   7X     SBM00940
```

```
              2       9HDESTROYED 10X 4HTYPE/16X 6HNUMBER , 7X                SBM00950
              3       9HCONDITION, 6X  8HDETECTED, 41X 4HSHIP/)               SBM00960
 FORM 3B     FORMAT (I19, I13, I15, 2L15, J19)                                SBM00970
                                                                              SBM00980
 FORM 3C     FORMAT (1H0 44X 29HRECORD OF SUCCESSFUL TRANSITS//               SBM00990
             1        40X 7HTRANSIT 6X 4HTYPE 14X 8HPOSITION/40X6HNUMBER     SBM01000
             2        7X 4HSHIP 12X 1HX 10X 1HY//)                           SBM01001
 FORM 3D     FORMAT (37X I6, 8X J6, 10X F3.1, F11.1)                          SBM01010
             PERMANENT EVENT STATUS                                           SBM01020
                                                                              SBM01030
             IF(BRAND NEW), STA BAR 4                                         SBM01040
             IF (BARRIER DEAD), STA BAR 1                                     SBM01050
             IF (LENGTH(SURF WEAPON) .LE. 0), STA BAR 2                       SBM01060
             IF (LENGTH(SUB WEAPON)  .LE. 0),STA BAR 3                        SBM01070
             IF (X .GE. 5), NEW TRANS 1                                       SBM01080
             IF (TRANS HIS(J,4)), NEW TRANS 2                                 SBM01090
             NEXT EVENT                                                       SBM01100
   FORM 1A   FORMAT(1H0)                                                      SBM01110
STA BAR 1    EXECUTE RECORD (3)                                               SBM01120
             BARRIER DEAD = FALSE                                             SBM01121
             GO TO STA BAR 4                                                  SBM01130
STA BAR 2    EXECUTE RECORD (7)                                               SBM01140
             GO TO STA BAR 4                                                  SBM01150
STA BAR 3    EXECUTE RECORD (8)                                               SBM01160
STA BAR 4    RESET LENGTH (SUB WEAPON) TO 22                                  SBM01170
             RESET LENGTH (SURF WEAPON) TO 3                                  SBM01180
             IF(BRAND NEW), FIRST THRU                                        SBM01190
             IF (X .GE. 5), NEW TRANS 1                                       SBM01200
             IF(TRANS HIS (J,4)),    NEW  TRANS 2                             SBM01210
                                                                              SBM01220
NEW TRANS 1  EXECUTE RECORD (6)                                               SBM01230
             PLACE (J, NAME(J),X, Y ) IN SUCCESSFUL TRANSIT                   SBM01240
             GO TO  NEW TRANS 3                                               SBM01250
NEW TRANS 2  EXECUTE RECORD(1)                                                SBM01260
             GO TO NEW TRANS 3                                                SBM01270
FIRST THRU   J=0                                                              SBM01280
             BRAND NEW =FALSE                    ...WIPE OUT THE TAG          SBM01290
```

```
NEW TRANS 3 CONDITION = RANDOM                                          SBM01300
           IF (CONDITION .LE. .25), SET 1                               SBM01310
           IF (CONDITION .LE. .5), SET 2                                SBM01320
           IF (CONDITION .LE. .75), SET 3                               SBM01330
   SET 4   SON STAT = 4                                                 SBM01340
           GO TO SET SONAR                                              SBM01350
  SET 1    SON STAT = 1                                                 SBM01360
           GO TO SET SONAR                                              SBM01370
  SET 2    SON STAT = 2                                                 SBM01380
           GO TO  SET SONAR                                             SBM01390
  SET 3    SON STAT =3                                                  SBM01400
                                                                        SBM01410
          XUPDATE CURRENT SONAR  CONDITIONS                             SBM01420
                                                                        SBM01430
SET SONAR  DO (JOB 1 ) UNTIL I .G. 4 , I                                SBM01440
   JOB 1   REPLACE (*  .NE. SON STAT ,,,,,,,,,,,I) BY ENTRY            SBM01450
          1       SON TAB(((SON STAT -1)* 4) + I) IN  CUR SON           SBM01460
START TRANS  J= J+1.        ... INCREMENT TRANSIT NUMBER                SBM01470
           Y = 1.8 + J*.2     ...LOCATE STARTING POINT                  SBM01480
            X= 1.0                                                      SBM01490
           LANE = TRUE                                                  SBM01491
           IF (Y .GE. 12),TERMINATE, PASS                               SBM01500
  TERMINATE MOVE TIME (1)= 0        ...CLOSE PATH TO ADVANCE            SBM01510
           ATTACK TIME(1)=0        ...CLOSE PATH TO ATTACK              SBM01520
           NEXT EVENT               ...EXIT TO WRAP UP OF DATA          SBM01530
          X     DETERMINE THE TYPE OF TRANSITOR                         SBM01540
     PASS  STAR=RANDOM                                                  SBM01550
           IF (STAR .LE. .6), TYPE 1                                    SBM01560
           IF (STAR .LE. .7), TYPE 2                                    SBM01570
           IF (STAR .LE. .8), TYPE 3                                    SBM01580
           IF (STAR .LE. .9), TYPE 4                                    SBM01590
  TYPE 5   TYPE = 5                                                     SBM01600
           CNAME= MER(1)                                                SBM01610
           GO TO HISTORY                                                SBM01620
  TYPE 1   TYPE = 1                                                     SBM01630
           CNAME=SUB(1)                                                 SBM01640
           GO TO HISTORY                                                SBM01650
```

```
 TYPE 2     TYPE = 2                                                        SBM01660
            CNAME=NUC (1)                                                   SBM01670
            GO TO HISTORY                                                   SBM01680
 TYPE 3     TYPE =3                                                         SBM01690
            CNAME=FSH(1)                                                    SBM01700
            GO TO HISTORY                                                   SBM01710
 TYPE 4     TYPE =4                                                         SBM01720
            CNAME=DDE (1)                                                   SBM01730
           XFILL OUT NEW ENTRY IN TRANSIT HISTORY LIST                      SBM01740
HISTORY     PLACE    (SON STAT, 0, FALSE, FALSE,    CNAME) IN TRANS HIS     SBM01750
            REPLACE ENTRY ADVANCE (1) BY (* ,CNAME, X, Y)                   SBM01760
            END                                                             SBM01770
            CONTINGENT EVENT ADVANCE ((MOVE TIME, MTYPE, CUR X,             SBM01780
           1                          CUR Y), 1)                           SBM01790
            PROGRAM OBJECT  MTYPE                                           SBM01800
            REAL ADVANCE                                                    SBM01810
            X = CURX (1)                                                    SBM01820
            Y = CURY (1)                                                    SBM01830
  JUMP IN   PROB = CUR SON (X,Y) * DETECT FACTOR (ATTACKER)                 SBM01840
            UNLESS (LANE .AND.(RANDOM .LE. PROB)), DIRECTION                SBM01850
           X    A DETECTION OCCURS ONLY WHEN BOTH TRANS IN THE LANE AND     SBM01860
           X    A SUFFICIENT PROBABILITY EXISTS                             SBM01870
DETECT      TIMES DETECTED (J) = TIMES DETECTED (J) + 1                     SBM01880
            IF((TIMES DETECTED (J).G.5).AND.(X.G.4)), WHO, DIRECTION        SBM01890
           XTHIS ANSWERS DO WE HAVE ENOUGH INFO AND IS HE CLOSE ENOUGH      SBM01900
 WHO        IF(CNAME .IN. ENEMY), TRY , DIRECTION  ...IS HE AN ENEMY        SBM01910
 TRY        IF (CNAME .IN. SUBMARINE) , UNDER    ...SELECT WEAPON TYPE      SBM01920
            WEAPON TYPE =SURF(1)                                            SBM01930
            GO TO HOW MANY                                                  SBM01940
 UNDER      WEAPON TYPE = SUBM (1)                                          SBM01950
           X          HOW MANY WEAPONS SHOULD  WE SHOOT                    SBM01960
HOW MANY    IF (CNAME .IN. DANGEROUS), MANY                                 SBM01970
 FEW        WEAPON NUMBER =1                                                SBM01980
            GO TO SET UP                                                    SBM01990
MANY        WEAPON NUMBER =3                                                SBM02000
           X       SET UP FOR AN ATTACK  EVENT                              SBM02010
 SET UP     REPLACE ENTRY ATTACK(1) BY (EPSILON (TIME), J ,WEAPON TYPE,     SBM02020
```

```
          1                                   WEAPON NUMBER)                    SBM02030
DIRECTION  HEAD = RANDOM                                                        SBM02040
           IF(HEAD .GE.  0.7), STEP 3                                           SBM02050
           IF(HEAD .GE.  0.4), STEP 2                                           SBM02060
 STEP 1    DX= .1*SPEED RATIO (TYPE)                                          SBM02070
           DY= 0                                                              SBM02080
           GO TO LANE CHECK                                                   SBM02090
 STEP 2    DX =.06* SPEED RATIO(TYPE)                                         SBM02100
           DY =.03* SPEED RATIO(TYPE)                                         SBM02110
           GO TO  LANE CHECK                                                  SBM02120
 STEP 3    DX =.06* SPEED  RATIO (TYPE)                                       SBM02130
           DY =-.03 * SPEED RATIO (TYPE)                                      SBM02140
LANE CHECK IF(Y+DY.G.12.OR.Y+DY.L.2), OUTSIDE                                 SBM02150
           LANE =TRUE                                                         SBM02160
           GO TO SET ONE                                                      SBM02170
OUTSIDE    LANE = FALSE                                                       SBM02180
            EXECUTE RECORD (5)                                                SBM02181
          X          SET UP FOR NEXT EVENT ADVANCE                            SBM02190
SET ONE    REPLACE ENTRY ADVANCE (1) BY (TIME+.1, *, X+DX, Y+DY)              SBM02200
           END                                                                SBM02210
           CONTINGENT EVENT ATTACK ((ATTACK TIME, ATRANSIT, ASSIGNED          SBM02220
          1            WEAPON TYPE, NUMBER WEAPS ASSIGNED), 1)                SBM02230
           INTEGER ATRANSIT, NUMBER WEAPS ASSIGNED, WEAPON. NUMBER            SBM02239
           PROGRAM OBJECT  ASSIGNED WEAPON TYPE                               SBM02240
           ATTACK TIME (INDEX) = 0                                            SBM02241
ATT 0      ATTACKED (J) = TRUE                                                SBM02250
ATT 1      IF(ASSIGNED WEAPON TYPE (1) .IN. SURF), ATT 3                      SBM02260
           ...THIS CHECKS TO SHOOT NO MORE THAN ARE ON BOARD                  SBM02270
ATT 2      UNLESS (LENGTH(SUB WEAPON) .LE. NUMBER WEAPS ASSIGNED (1)),        SBM02280
          1          ATT 2B                                                  SBM02290
ATT 2A     RESET LENGTH (SUB WEAPON) TO 0     ...SHOOT ONLY WHAT IS LEFTSBM02300
           GO TO EFFECT                                                       SBM02310
ATT 2B     RESET LENGTH(SUB WEAPON) TO LENGTH (SUB WEAPON) - NUMBER           SBM02320
          1            WEAPS ASSIGNED (1)     ...SHOOT WHAT  WE NEED        SBM02330
           GO TO EFFECT                                                       SBM02340
ATT 3      UNLESS(LENGTH(SURF WEAPON) .LE. NUMBER WEAPS ASSIGNED (1)),        SBM02350
          1          ATT 3B                                                  SBM02360
```

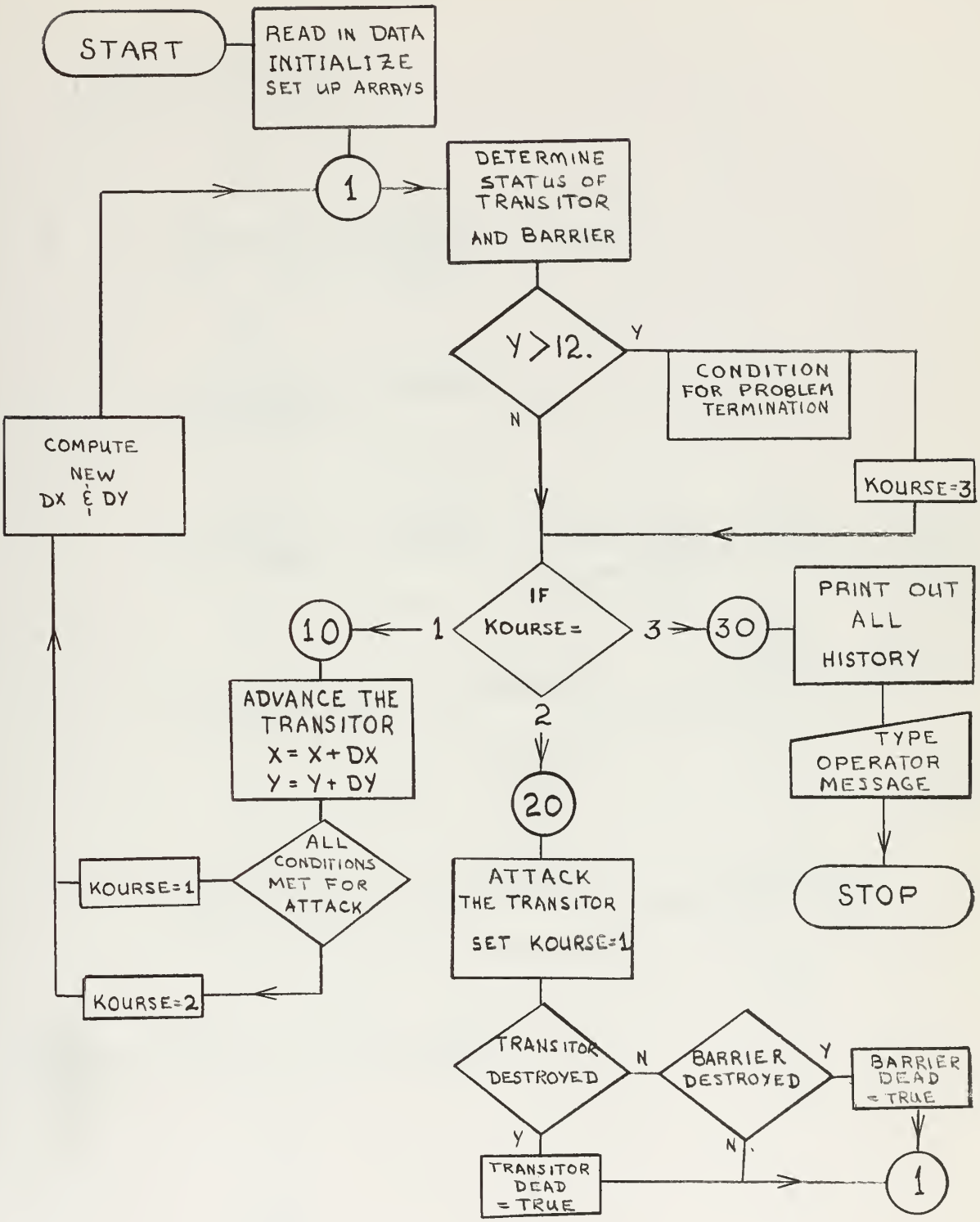
```
ATT 3A      RESET LENGTH (SURF WEAPON) TO 0  ...SHOOT ONLY WHAT IS LEFT SBM02370
            GO TO EFFECT                                               SBM02380
ATT 3B      RESET LENGTH (SURF WEAPON) TO LENGTH(SURF WEAPON) - NUMBER  SBM02390
           1                WEAPS ASSIGNED (1)  ...SHOOT ALL WE NEED   SBM02400
EFFECT      IF (RANDOM .L.(.9*KILL FACTOR(TYPE))), SINK ...HIT OR MISS  SBM02410
  NO SINK   IF (RANDOM .L.  .4),  SECOND C                             SBM02420
 RETALIATE  IF(RANDOM .G.  .6), KOBARR                                 SBM02430
           X  SHOULD ABOVE BE FALSE THE TRANSITOR DOES NOT  TRY TO     SBM02440
           Y  SHOOT BACK BUT IS ASSURED OF A SUCCESSFUL TRANSIT        SBM02450
           Z  OTHERWISE HE  SHOOTS AND DESTROYS THE BARRIER SUB        SBM02460
SLIPAWAY    EXECUTE RECORD (4)                                         SBM02470
            X=5                                                        SBM02480
            NEXT EVENT      ...NORMAL EXIT FOR SHOT AT BUT MISSED      SBM02490
           X                    AND NO CHANCE FOR ANOTHER ATTACK       SBM02500
KOBARR      BARRIER DEAD = TRUE                                        SBM02510
            NEXT EVENT      ...NORMAL EXIT BARRIER  DESTROYED          SBM02520
SECOND C    EXECUTE RECORD (2)                                         SBM02530
            NEXT EVENT      ...NORMAL EXIT FOR SHOT AT BUT MISSED      SBM02540
           X                   BUT SOME CHANCE FOR ANOTHER SHOT        SBM02550
 SINK       TRANS HIS (J,4) = TRUE                                     SBM02560
            END                                                        SBM02570
             SUSPEND FAP LISTING
            END COMPILATION                                            SBM02580
```

```
              X PROCEDURE RECORD                                                    SBR00040
                                                                                    SBR00050
                COMMON    ...BLOCK A...                                           SBR00060
              1PROBLEM NUMBER,PAGE, LINE, IN, OUT                                   SBR00070
                    INTEGER PROBLEM NUMBER, PAGE, LINE, IN, OUT                     SBR00080
                COMMON    ...BLOCK B...                                           SBR00090
              1EVENT NUMBER, TYPE, NEXT EVENT, J, X, Y, TRANS HIS,LLTHIS            SBR00100
                    LIST TRANS HIS ((SONAR COND, TIMES DETECTED, ATTACKED,          SBR00110
              1                     DESTROYED, NAME), LLTHIS)                     SBR00120
                    PROGRAM OBJECT NAME                                             SBR00130
                    LOGICAL ATTACKED,DESTROYED, TRANS HIS                           SBR00140
                    INTEGER TYPE, TIMES DETECTED, EVENT NUMBER,J                    SBR00150
                    INTEGER SONAR COND                                              SBR00151
                                                                                    SBR00160
                                                                                    SBR00170
60            PROCEDURE RECORD (EVENT TYPE)                                         SBR00180
              INTEGER EVENT TYPE                                                    SBR00190
              INTEGER ALFA                                                          SBR00200
   NEW        EXECUTE TITLE (FORM H)                                               SBR00210
   OLD 1      IF(TYPE .G.  1), OLD 2                                               SBR00220
              ALFA = 3HSUB                                                          SBR00230
              GO TO WHICH EVENT                                                     SBR00240
   OLD 2      IF (TYPE .G.  2), OLD 3                                              SBR00250
              ALFA = 3HNUC                                                          SBR00260
              GO TO WHICH EVENT                                                     SBR00270
   OLD 3      IF (TYPE .G.  3), OLD 4                                              SBR00280
              ALFA = 3HFSH                                                          SBR00290
              GO TO WHICH EVENT                                                     SBR00300
   OLD 4      IF(TYPE .G. 4),  OLD 5                                               SBR00310
              ALFA = 3HDDE                                                          SBR00320
              GO TO WHICH EVENT                                                     SBR00330
   OLD 5      ALFA = 3HMER                                                         SBR00340
WHICH EVENT   IF (EVENT TYPE .LE. 1), RECORD 1   ...TRANSITOR SUNK                 SBR00350
              IF (EVENT TYPE .LE. 2), RECORD 2   ...ABORT ATTACK REATTACK         SBR00360
              IF (EVENT TYPE .LE. 3), RECORD 3   ...KO BARRIER                    SBR00370
              IF (EVENT TYPE .LE. 4), RECORD 4   ...ABORT ATTACK FREETRANS        SBR00380
              IF (EVENT TYPE .LE. 5), RECORD 5   ...OUT OF BOUNDS                 SBR00390
```

```
                IF (EVENT TYPE .LE. 6), RECORD 6    ...SAFE TRANSIT              SBR00400
                IF (EVENT TYPE .LE. 7), RECORD 7    ...SURFACE WEAPS DEPLETEDSBR00410
                IF (EVENT TYPE .LE. 8), RECORD 8    ...SUBM WEAPS DEPLETED       SBR00420
RECORD 1        ...TRANSITOR SUNK BY BARRIER                    ...CONTINUE      SBR00430
                WRITE (OUT,FORM 1) EVENT NUMBER +1 , TIME, J, X, Y, ALFA         SBR00440
                GO TO FIX                                                        SBR00450
RECORD 2        ...ATTACK ABORTED BUT REATTACK POSSIBLE         ...CONTINUE      SBR00460
                WRITE(OUT,FORM 2) EVENT NUMBER +1, TIME, J, X, Y, ALFA           SBR00470
                GO TO FIX                                                        SBR00480
RECORD 3        ...BARRIER KOD BY TRANSITOR                     ...CONTINUE      SBR00490
                WRITE (OUT,FORM 3) EVENT NUMBER +1, TIME, J, X, Y, ALFA          SBR00500
                GO TO FIX                                                        SBR00510
RECORD 4        ...ATTACK ABORTED, NO REATTACK POSSIBLE         ...CONTINUE      SBR00520
                WRITE (OUT, FORM 4) EVENT NUMBER +1,TIME, J, X, Y, ALFA          SBR00530
                GO TO FIX                                                        SBR00540
RECORD 5        ...TRANSITOR OUT OF LANE                        ...CONTINUE      SBR00550
                WRITE (OUT, FORM 5) EVENT NUMBER +1, TIME, J, X, Y, ALFA         SBR00560
                GO TO FIX                                                        SBR00570
RECORD 6        ...SAFE TRANSIT                                 ...CONTINUE      SBR00580
                WRITE (OUT, FORM 6) EVENT NUMBER + 1 , TIME, J, X , Y, ALFA,SBR00590
               1         TIMES DETECTED (J)                                      SBR00600
                GO TO FIX                                                        SBR00610
RECORD 7        ...SURFACE WEAPONS DEPLETED                     ...CONTINUE      SBR00620
                WRITE (OUT, FORM 7) EVENT NUMBER +1, TIME, J, X, Y, ALFA         SBR00630
                GO TO FIX                                                        SBR00640
RECORD 8        ...SUBMARINE WEAPONS DEPLETED                   ...CONTINUE      SBR00650
                WRITE (OUT, FORM 8) EVENT NUMBER +1, TIME, J, X, Y, ALFA         SBR00660
                GO TO FIX                                                        SBR00670
        FIX     EVENT NUMBER = EVENT NUMBER + 1                                  SBR00680
                LINE = LINE - 2                                                  SBR00690
                RETURN          ...NORMAL EXIT FROM PROCEDURE                    SBR00700
FORM H          FORMAT (13HOEVENT   TIME4X 7HTRANSIT 10X 15HSHIP POSITION       SBR00710
               1         4HTYPE 4X 5HEVENT/7H NUMBER 10X 8HINVOLVED 11X 1HX      SBR00720
               2         7X 1HY 4X 4HSHIP 4X 11HDESCRIPTION //)                  SBR00730
 FORM 1         FORMAT (I5, 3X, F5.1,4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,       SBR00740
               1         4X, 14HTRANSITOR SUNK//)                               SBR00750
 FORM 2         FORMAT (I5, 3X, F5.1,4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,       SBR00760
```

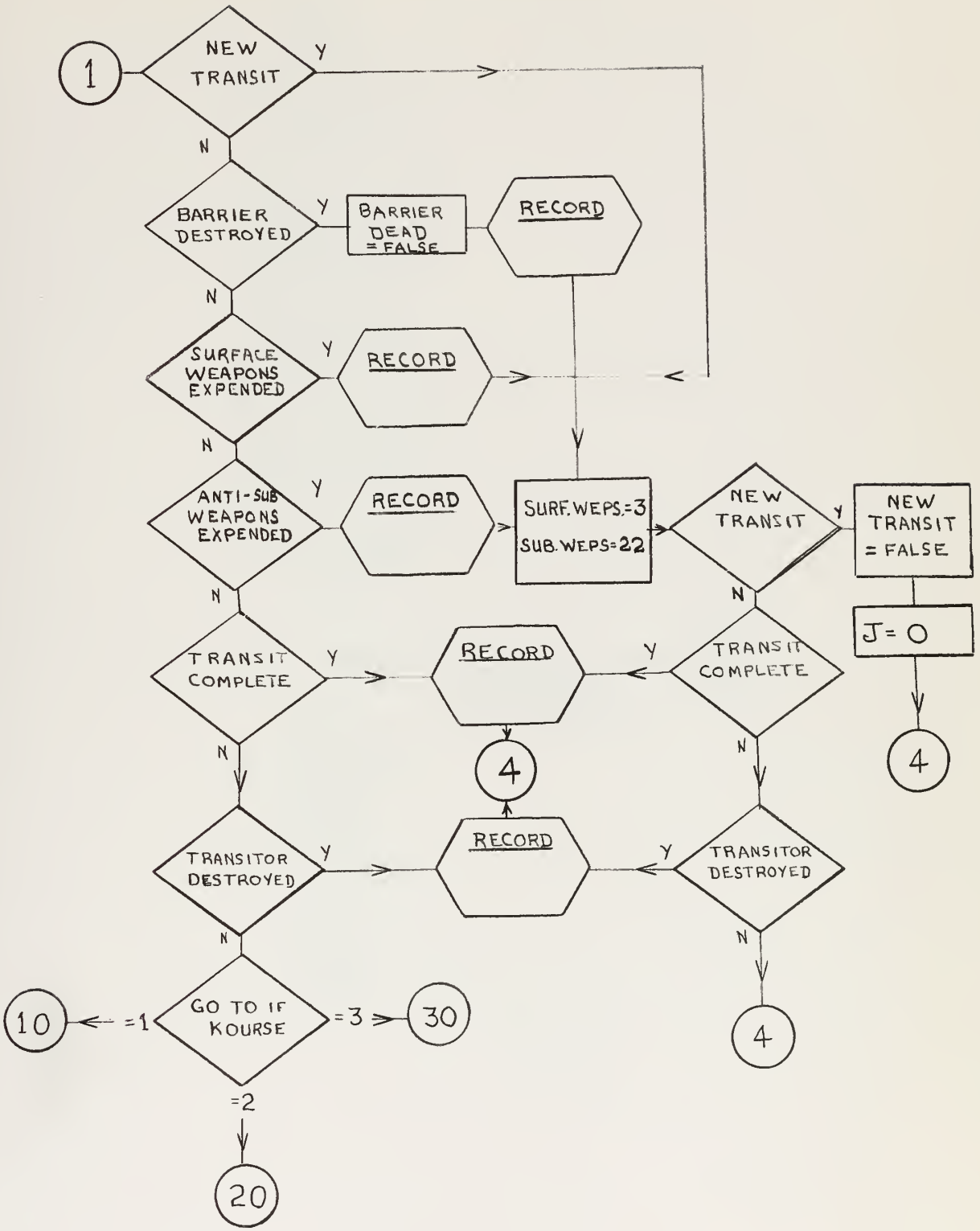
```
     1        4X, 38HATTACK ABORTED DUE TO SSK MATL FAILURE/ 57X, SBR00770
     2        17HREATTACK POSSIBLE)                              SBR00780
FORM 3   FORMAT (I5, 3X, F5.1,4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00790
     1        4X, 45HBARRIER SSK DESTROYED AFTER HIS ATTACK MISSEDSBR00800
     2        //)                                                 SBR00810
FORM 4   FORMAT (I5, 3X, F5.1,4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00820
     1       4X, 38HATTACK ABORTED DUE TO SSK MATL FAILURE  /57X, SBR00830
     2       21HNO CHANCE TO REATTACK)                           SBR00840
FORM 5   FORMAT (I5, 3X, F5.1, 4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00850
     1       4X,  31HTRANSITOR OUTSIDE LANE BOUNDARY//)          SBR00860
FORM 6   FORMAT (I5, 3X, F5.1, 4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00870
     1       4X, 23HSAFE TRANSIT, NO DAMAGE / 57X 4HONLY I3,     SBR00880
     2       11H DETECTIONS)                                     SBR00890
FORM 7   FORMAT (I5, 3X, F5.1, 4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00900
     1       4X, 33HALL ANTI-SURFACE WEAPONS DEPLETED//)         SBR00910
FORM 8   FORMAT (I5, 3X, F5.1, 4X, I4, 12X, F6.2, 2X, F6.2, 3X, A3,  SBR00920
     1       4X, 35HALL ANTI-SUBMARINE WEAPONS DEPLETED//)       SBR00930
      SUSPEND FAP LISTING
      END COMPILATION                                            SBR00940
```

```
                ...PROCEDURE TITLE                                                  SBT00040
                                                                                    SBT00050
                COMMON...BLOCK A...                                                 SBT00060
               1PROBLEM NUMBER, PAGE, LINE, IN, OUT                                 SBT00070
                INTEGER PROBLEM NUMBER, PAGE, LINE, IN, OUT                         SBT00080
                PROCEDURE TITLE (DATA HEADING)                                      SBT00090
                NORMAL MODE INTEGER                                                 SBT00100
                UNLESS(LINE .LE. 0), RETURN     ...CHECK FOR TOP OF PAGE            SBT00110
                LINE = 50                       ...RESET FOR FULL PAGE              SBT00120
                PAGE = PAGE + 1                                                     SBT00130
                WRITE (OUT, PAGE HEAD) PROBLEM NUMBER, PAGE                         SBT00140
                WRITE(OUT, DATA HEADING)                                            SBT00150
  RETURN        RETURN          ... NORMAL EXIT                                   SBT00160
  PAGE HEAD     FORMAT (15H1PROBLEM NUMBER I6, 25X                                SBT00170
                1         30HSINGLE SUBMARINE BARRIER MODEL 34X 4HPAGE I5)        SBT00180
                  SUSPEND FAP LISTING
                END COMPILATION                                                     SBT00190
```


START
READ IN DATA
INITIALIZE
SET UP ARRAYS
1
DETERMINE
STATUS OF
TRANSITOR
AND BARRIER
Y>12.
Y
CONDITION
FOR PROBLEM
TERMINATION
N
KOURSE=3
COMPUTE
NEW
DX & DY
IF
KOURSE=
1
10
3
30
PRINT OUT
ALL
HISTORY
2
20
ADVANCE THE
TRANSITOR
X = X+DX
Y = Y+DY
TYPE
OPERATOR
MESSAGE
ALL
CONDITIONS
MET FOR
ATTACK
KOURSE=1
ATTACK
THE TRANSITOR
SET KOURSE=1
STOP
KOURSE=2
TRANSITOR
DESTROYED
N
BARRIER
DESTROYED
Y
BARRIER
DEAD
= TRUE
Y
N
TRANSITOR
DEAD
= TRUE
1

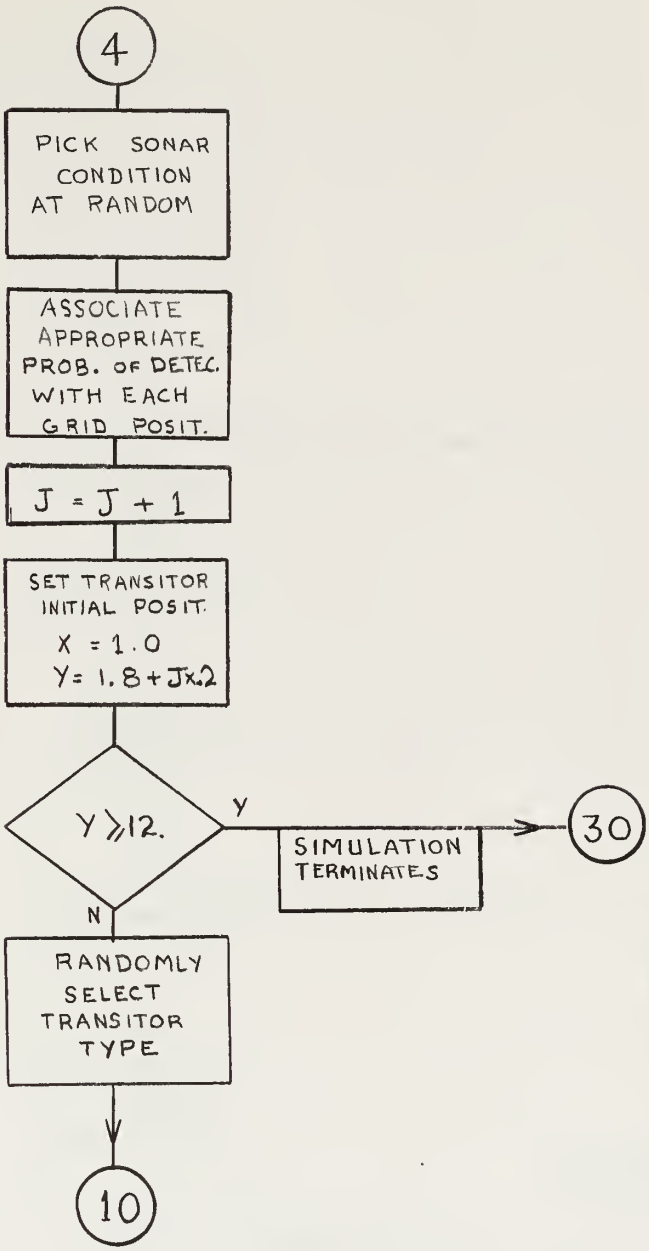
FORTRAN 63 MAIN PROGRAM

CHART 1



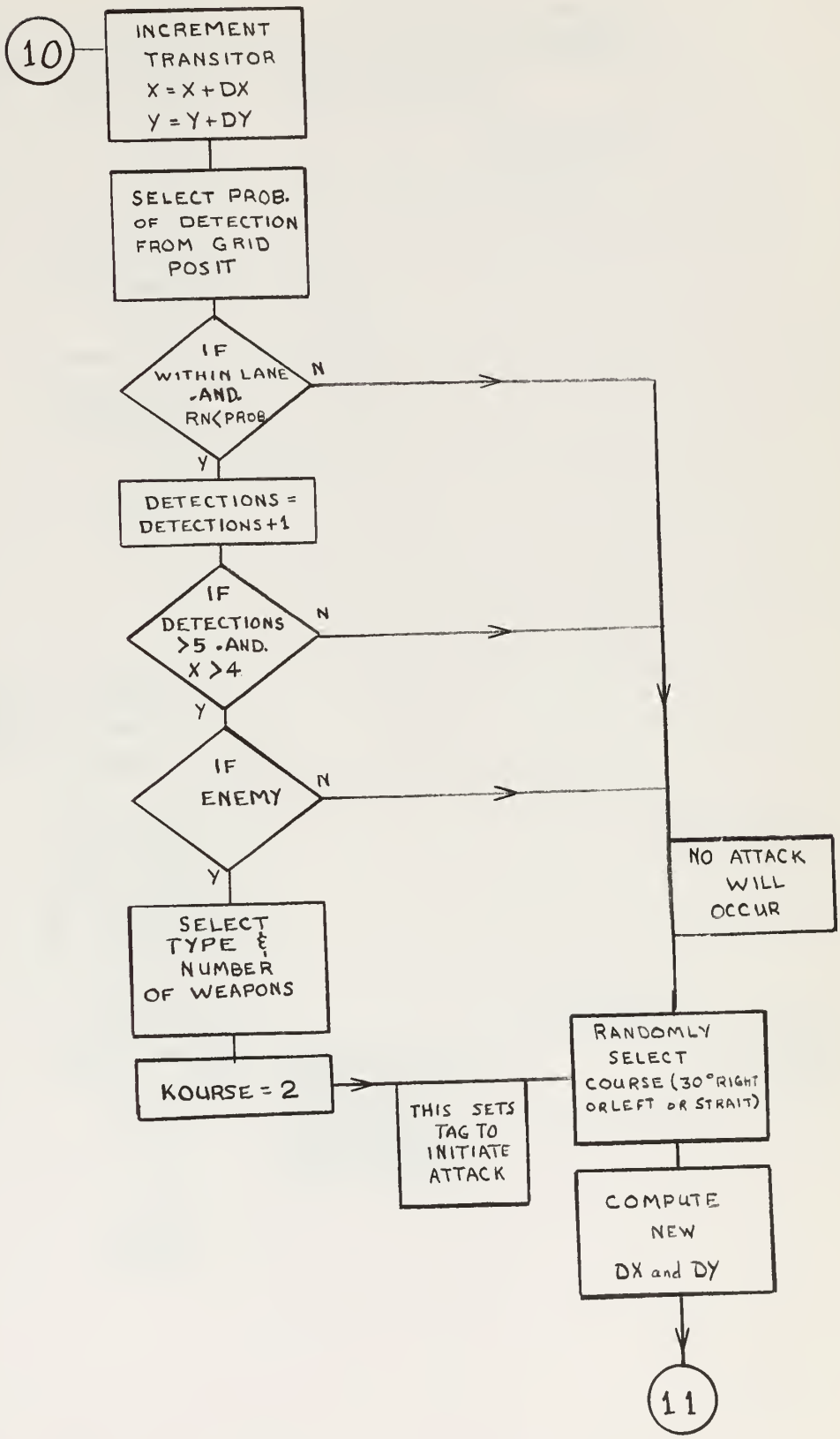
FORTRAN 63 MAIN PROGRAM

CHART 2



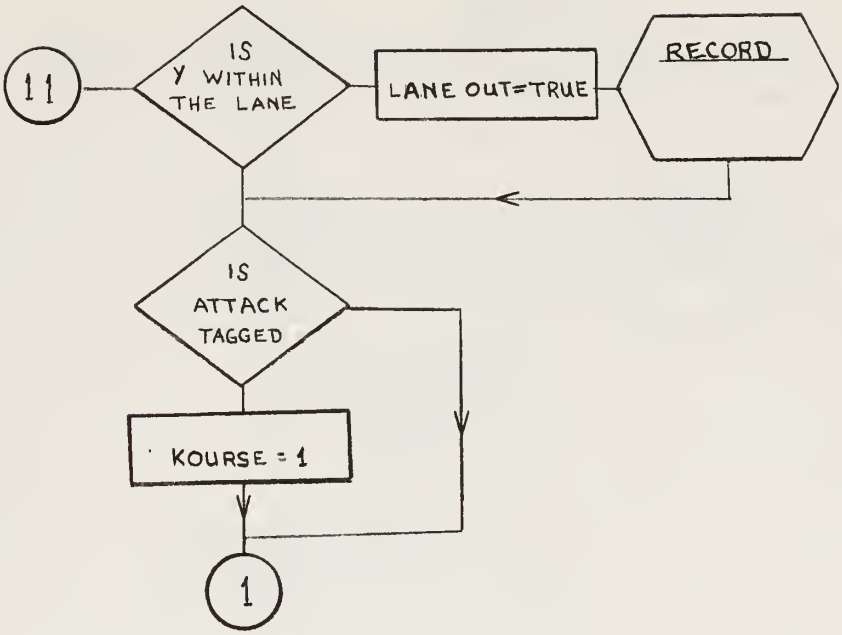
FORTRAN 63 MAIN PROGRAM

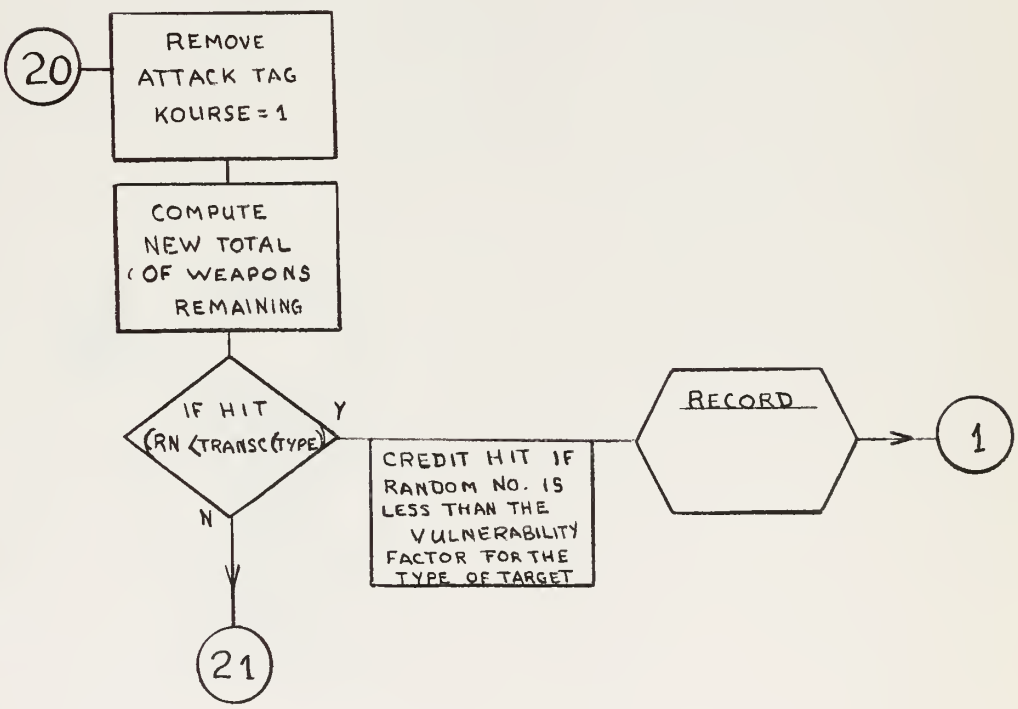
CHART 3



FORTRAN 63 MAIN PROGRAM

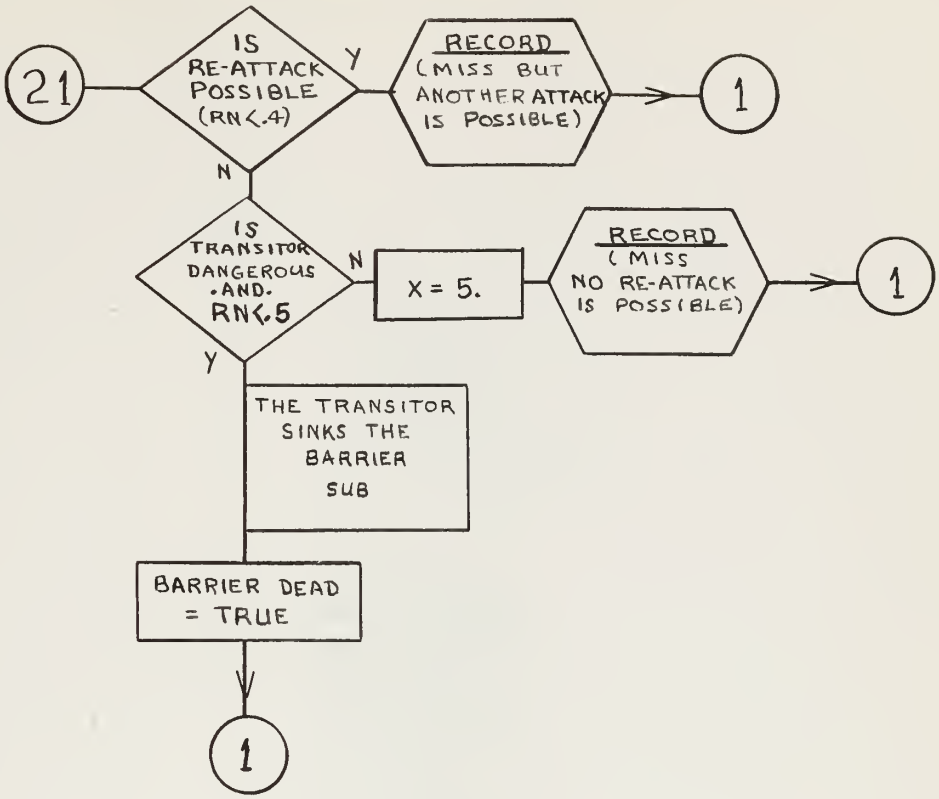
CHART 4

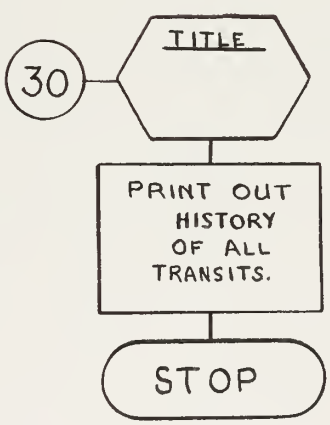




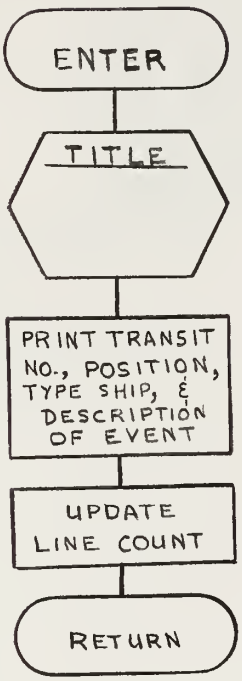
FORTRAN 63 MAIN PROGRAM

CHART 5

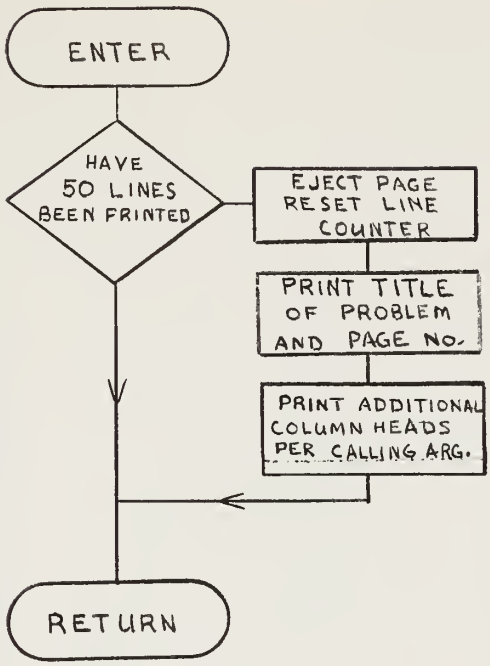




FORTRAN 63 SUBROUTINE RECORD



FORTRAN 63 SUBROUTINE TITLE


ENTER
HAVE 50 LINES BEEN PRINTED
EJECT PAGE RESET LINE COUNTER
PRINT TITLE OF PROBLEM AND PAGE NO.
PRINT ADDITIONAL COLUMN HEADS PER CALLING ARG.
RETURN

```
      PROGRAM SUBBAR 1
      COMMON /BLKA/ IPROB, IPAGE, LINE
      COMMON /BLKB/ J, NAM, X, Y, INVENTO
      COMMON/ BLKC/IHIST
      DIMENSION TRANSC(5,4)
      DATA(TRANSC=1.,2.,3.,4.,5.,1.,3.,.5,4.,4.,1.,.8,1.,1.,1.,.9,.9,
     1            1.,1.,1.)
      TYPE LOGICAL LAOUT, LBNEW, LBDEAD,LTAG,LHIST, LCLASS
      DIMENSION LHIST(100, 2),IHIST(100,3), ITRSUC(100),CURSON(4,10).
     1          SONTAB(4,10,4)
      DIMENSION SYPE (5)
      DATA(SYPE = 3HSUB, 3HNUC, 3HFSH, 3HDDE, 3HMER)
      DIMENSION LCLASS(5,4)
      DATA(LCLASS =7773260000000000B)
      READ 9002, IPROB
      IPAGE = 0
      LINE=0
      LAOUT=0
      LBNEW=1
      LBDEAD=0
      LTAG=0
      DO 101 K=1,100
      LHIST (K,1)=0
  101 LHIST(K,2)=0
      INVENTO=0
      READ 9003,   (((SONTAB(K,L,M),L=1,10),K=1,4),M=1,4)
      CALL TITLE (1)
      PRINT 9004
      PRINT 9005, ((M,(SONTAB(K,L,M),L=1,10),K=1,4),M=1,4)
 9002 FORMAT(I5)
 9003 FORMAT(10F 6.2)
      LINE=0
      CALL TITLE(1)
      PRINT 9006
      PRINT 9007
      PRINT 9008,     (SYPE (K),(TRANSC (K,M), M=1,4), K=1,5)
      LINE=0
```

```
      READ 9009, INITIL
      PRINT 9010,INITIL
      PRINT 9013
      PRINT 9014, (SYPE (N),(LCLASS (N,M), M=1,4),N=1,5)
 9004 FORMAT(46X,25HPOSSIBLE SONAR CONDITIONS/ 49X, 11HSONAR TABLE
     1         8H(SONTAB))
 9005 FORMAT( 7H0ISOSTA, I3,5X, 10F10.2)
 9006 FORMAT(40X, 37HCHARACTERISTIC OF TRANSITORS (TRANSC)///)
 9007 FORMAT(38X, 4HTYPE 7X,4HKIND, 6X,5HSPEED, 5X,4HKILL, 5X,
     1       6HDETECT/38X, 4HSHIP, 17X,5HRATIO, 4X,6HFACTOR,4X,
     2       6HFACTOR)
 9008 FORMAT(1H038X,A3,    4F10.1 )
 9009 FORMAT (I15)
 9010 FORMAT(/////39HINITIALIZING VALUE FOR RANDOM NUMBER =  ,I15)
 9011 FORMAT(47X,23HHISTORY OF ALL TRANSITS// 14X, 7HTRANSIT, 10X
     1         5HSONAR, 12X, 5HTIMES, 14X, 8HATTACKED, 8X, 9HDESTROYED,
     2         9X, 4HTYPE/14X,6HNUMBER,11X,9HCONDITION,8X,8HDETECTED)
 9012 FORMAT( 16X, I2,15X, I2, 15X, I3,15X,L1,16X, L1, 15X, A3)
 9013 FORMAT(////52X 16HCLASS MEMBERSHIP//54X 12HARRAY LCLASS//37X
     1       4HTYPE, 7X, 5HCLASS ,5X, 5HCLASS,5X,5HCLASS,5X,5HCLASS/
     2       37X,4HSHIP, 7X, 3HALL, 7X,5HENEMY, 5X,9HDANGEROUS
     3       10H SUBMARINE)
 9014 FORMAT(37X, A3,  4L10 )
  302 IF(LBNEW) 322,303
  303 IF(LBDEAD)315,304
  304 IF(ISURFW .LE. 0) 318,305
  305 IF(ISUBWE .LE. 0) 321,306
  306 IF(X .GE. 5.)330,307
  307 IF(LHIST(J,2))333,308
  308  CONTINUE
      GO TO(1000, 2000,3000) KOURSE
C     THIS ALLOWS BRANCHING TO CONTINGENT EVENTS OR TERMINATION
  315 CALL RECORD (3)
      LBDEAD=0
      GO TO 322
  318 CALL RECORD(7)
      GO TO 322
```

```
  321 CALL RECORD (8)
  322 ISUBWE=22
      ISURFW=3
      IF(LBNEW) 335, 325
  325 IF(X.GE.5.0) 330,326
  326 IF(LHIST(J,2)) 333,330
  330 CALL RECORD (6)
      ITM=ITM+1
      ITRSUC(ITM)=J
      GO TO 338
  333 CALL RECORD (1)
      GO TO 338
  335 J=0
      LBNEW=0
C        GENERATE AN INTEGER SONAR CONDITION (ISOSTA)
  338 SONRAND=RANF(INITIL)
      IF (SONRAND .LE. .25)    3390,3381
 3381 IF (SONRAND .LE. .5) 3391,3382
 3382 IF (SONRAND .LE. .75) 3392,3384
 3384 ISOSTA = 4
      GO TO 3300
 3390 ISOSTA = 1
      GO TO 3300
 3391 ISOSTA =2
      GO TO 3300
 3392 ISOSTA = 3
 3300 DO 339 L=1,10
      DO 339 K=1,4
C         UPDATE CURRENT SONAR CONDITIONS OVER GRID
  339 CURSON (K,L) = SONTAB(K,L,ISOSTA)
      J=J+1
      IHIST (J,2) = 0
      IHIST(J,1)=ISOSTA
      Y=1.8+J*.2
      X=1.0
      LAOUT=0
      IF(Y .GE. 12.) 3000, 343
```

```
C      TERMINATION OF SIMULATION OCCURS AT LABEL 3000
 3000 CONTINUE
      LINE=0
      CALL TITLE(3)
      PRINT 9011
      DO 3003   N=1,J
      NAM=IHIST(N,3)
 3003 PRINT 9012, (N, IHIST (N,1) , IHIST (N,2), LHIST(N,1),LHIST
     1              (N,2),SYPE(NAM))
      IDUMMY = ITYPE3(IDUMMY)
      STOP
C        DETERMINE TYPE OF TRANSITOR (NAM)
  343 TYPRAND = RANF(INITIL)
      IF (TYPRAND .LE. .6) 3531,3432
 3432 IF (TYPRAND .LE. .7) 3532,3433
 3433 IF (TYPRAND .LE. .8) 3533,3434
 3434 IF (TYPRAND .LE. .9) 3534,3435
 3435 NAM = 5
      GO TO 3400
 3531 NAM = 1
      GO TO 3400
 3532 NAM = 2
      GO TO 3400
 3533 NAM = 3
      GO TO 3400
 3534 NAM = 4
 3400 DX = 0.
      DY=0.
      IHIST(J,3) = NAM
C      THE HISTORY OF TRANSITOR TYPES (BY NUMBER) IS PART OF IHIST
C
C                ADVANCE
C
 1000 X=X+DX
      Y = Y + DY
      IX = X
      IY = Y -1.0
```

```
      PROB = CURSON(IX,IY) * TRANSC (NAM,4)
      IF(.NOT.LAOUT .AND. RANF(INITIL) .LE. PROB) 1100,1200
C          A DETECTION SENDS PROG TO 1100.
 1100 IHIST(J,2) =IHIST(J,2)+1
      IF(IHIST(J,2).GT. 5 .AND. X .GT. 4.0) 1120,1200
C     THIS ANSWER IS HE CLOSE ENOUGH AND DO WE HAVE ENOUGH INFO
 1120 IF(LCLASS(NAM,2))1121,1200
C     IS HE ENEMY   IF NOT GO TO LABEL 2000
 1121 IF(LCLASS (NAM,4)) 1122, 1124
C     IS HE SUBMARINE
 1122 IWEPTY = 2
      GO TO 1126
 1124 IWEPTY=1
C     ANTI SURFACE WEAPONS ARE SELECTED BY 1, ANTI SUB WEAPONS BY 2
C     NOW DETERMINE HOW MANY WEAPONS TO EXPEND
 1126 IF(LCLASS (NAM, 3))1127, 1128
 1127 IXPEND = 3
      GO TO 1129
 1128 IXPEND = 1
C     AN ATTACK MUST NOW BE TAGGED TO TAKE PLACE
 1129 KOURSE = 2
C     ALSO SET A TAG
      LTAG=1
 1200 HEAD=RANF(INITIL)
      N=NAM
      IF(HEAD .LE. 0.4) 1207,1201
 1201 IF(HEAD .LE. 0.7) 1205,1202
 1202 DX=.1* TRANSC(N,2)
 1203 DY=0.
      GO TO 1220
 1205 DX=.06*TRANSC(N,2)
      DY=.03*TRANSC(N,2)
      GO TO 1220
 1207 DX=.06*TRANSC(N,2)
      DY= -.03*TRANSC(N,2)
 1220 LAOUT = Y    .GT. 12.0 .OR. Y        .LT. 2.0
C          LAOUT IS SET TO TRUE WHEN TRANSITOR IS OUTSIDE THE LANE
```

```
C          NEXT CHECK TO SEE IF ATTACK HAS BEEN TAGGED
C          IF NOT ADVANCE IS TAGGED AND CONTROL GOES BACK TO PERMANENT
C          EVENT TO CHECK STATUS.
      IF(LTAG) 1223,1221
 1221 KOURSE=1
 1223 LTAG=0
      IF(LAOUT)1224,302
 1224 CALL RECORD(5)
 1225 GO TO 302
C
C          ATTACK
C
 2000 LHIST (J,1)=1
      KOURSE = 1
      IF(IWEPTY .EQ. 1) 2008, 2001
 2001 IF(IXPEND .GT. ISUBWE) 2002,2003
 2002 ISUBWE=0
      GO TO 2020
 2003 ISUBWE=ISUBWE - IXPEND
      GO TO 2020
 2008 IF(IXPEND .GT. ISURFW) 2009, 2010
 2009 ISURFW = 0
      GO TO 2020
 2010 ISURFW =ISURFW - IXPEND
C          EXPEND NUMBER OF WEAPONS ASSIGNED OR IF LESS ABOARD,ALL
C          THOSE REMAINING
 2020 TPROB=RANF(INITIL)
      IF(TPROB .LT.  .9*TRANSC(NAM, 3 )) 2100, 2021
C          WAS TRANSITOR DESTROYED  IF NOT WILL THERE BE ANOTHER CHANCE
C          TO TRY TO SHOOT HIM
 2021 IF(RANF(INITIL) .LT. .4) 2080,2022
C          IF THERE WAS A MISS CAN THE TRANSITOR KILL THE BARRIER
 2022 IF(LCLASS(NAM, 3) .AND. RANF(INITIL) .LT. .5) 2060, 2023
 2023 CALL RECORD (4)
C          THE TRANSITOR WAS UNSUCCESSFULLY ATTACKED , COULD NOT BE
C          REATTACKED, DID NOT ATTACK THE BARRIER SUB, BUT IS ASSURED
C          OF A SAFE TRANSIT
```

```
      X=5.
      GO TO 302
 2060 LRDEAD=1
C          THE TRANSITOP SUNK THE BARRIER SUB
      GO TO 302
 2080 CALL RECORD (2)
C          THE BARRIER MISSED BUT WILL HAVE ANOTHER CHANCE
      GO TO 302
 2100 LHIST(J,2)=1
C          THE TRANSITOR WAS DESTROYED
      GO TO 302
         END
```

```
      FUNCTION ITYPE3 (IDUMMY)
C     PRODUCES ON LINE TYPEWRITER MESSAGE
      DIMENSION IOUT(3)
      DATA(IOUT=8H   ALL T, 8HRANSITS , 8HCOMPLETE)

       WRITE (45,1)IOUT
    1 FORMAT(/ 3A8)
      ITYPE3 = IDUMMY
      RETURN
      END
```

```
      SUBROUTINE RECORD (IRECTY)
      COMMON /BLKA/ IPROB,IPAGE,LINE
      COMMON /BLKB/ J,NAM,X,Y,IVENTO
      COMMON /BLKC/ IHIST
      DIMENSION IHIST (100,3)
      CALL TITLE (2)
C         CHECK FOR TOP OF PAGE
      IVENTO = IVENTO + 1
      GO TO (810,820,830,840,850) NAM
  810 IALFA =3HSUB
      GO TO 852
  820 IALFA =3HNUC
      GO TO 852
  830 IALFA =3HFSH
      GO TO 852
  840 IALFA =3HDDE
      GO TO 852
  850 IALFA =3HMER
  852 GO TO (854, 856, 858, 860, 862, 864, 866, 868) IRECTY
  854 PRINT 9854, IVENTO  ,J, X, Y, IALFA
      GO TO 870
  856 PRINT 9856, IVENTO    ,J, X, Y, IALFA
      GO TO 870
  858 PRINT 9858, IVENTO    , J, X, Y, IALFA
      GO TO 870
  860 PRINT 9860, IVENTO    , J, X, Y, IALFA
      GO TO 870
  862 PRINT 9862, IVENTO    , J, X, Y, IALFA
      GO TO 870
  864 PRINT 9864, IVENTO    , J, X, Y, IALFA, IHIST (J,2)
      GO TO 870
  866 PRINT 9866, IVENTO    , J, X, Y, IALFA
      GO TO 870
  868 PRINT 9868, IVENTO    , J, X, Y, IALFA
  870 LINE = LINE - 2
      RETURN
 9854 FORMAT (I5, 12X, I4, 12X, F6.2, 2X,F6.2, 3X, A3, 4X
```

```
     1         14HTRANSITOR SUNK//)
9856 FORMAT (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         38HATTACK ABORTED DUE TO SSK MATL FAILURE/ 57X,
     2         17HREATTACK POSSIBLE)
9858 FORMAT  (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         45HBARRIER SSK DESTROYED AFTER HIS ATTACK MISSED//)
9860 FORMAT  (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         38HATTACK ABORTED DUE TO SSK MATL FAILURE / 57X,
     2         21HNO CHANCE TO REATTACK
9862 FORMAT  (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         31HTRANSITOR OUTSIDE LANE BOUNDARY//)
9864 FORMAT  (I5, 12X, I4, 12X, F6.2, 2X, F6.2,3X, A3, 4X
     1         23HSAFE TRANSIT, NO DAMAGE/ 57X 4HONLY I3, 2X,
     2         10HDETECTIONS)
9866 FORMAT (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         33HALL ANTI-SURFACE WEAPONS DEPLETED//)
9868 FORMAT (I5, 12X, I4, 12X, F6.2, 2X, F6.2, 3X, A3, 4X
     1         35HALL ANTI-SUBMARINE WEAPONS DEPLETED//)
         END
```

```
      SUBROUTINE TITLE(IREA)
      COMMON /BLKA/ IPROB,IPAGE,LINE
      IF(LINE) 10,10,400
C     THIS CHECKS FOR TOP OF PAGE
   10 LINE=50
      IPAGE=IPAGE+1
      PRINT 100,IPROB,IPAGE
 1000FORMAT (15H1PROBLEM NUMBER   I6, 23X, 18HFORTRAN SUBMARINE
     113HBARRIER MODEL  34X   4HPAGE  I5//)
      GO TO (101,201,203,205),IREA
  201 PRINT 110
  110  FORMAT(51X, 17HHISTORY OF EVENTS//2X5HEVENT 10X, 7HTRANSIT
     1       12X, 1HX, 7X, 1HY, 4X, 4HTYPE, 4X, 11HDESCRIPTION /
     2   2X, 6HNUMBER,  9X, 6HNUMBER, 26X, 4HSHIP, 4X, 8HOF EVENT)
      GO TO 400
  101 PRINT 9101
 9101 FORMAT (47X , 24HASSUMED CHARACTERISTICS ///)
      GO TO 400
  203 PRINT 9203
 9203 FORMAT(46X, 26HADDITIONAL SIMULATION DATA ///)
      GO TO 400
  205 CONTINUE
  400 CONTINUE
      RETURN
         END
```

APPENDIX III

INSTRUCTIONS FOR WRITING ON-LINE SIMULATIONS

This appendix contains detailed instructions for writing an on-line simulation employing the USNPGS satellite system. It is intended primarily for the reader with little or no background in assembly language programming. The desire is to make it possible for the user interested primarily in the quality of the simulation to employ the satellite system as his tool, without expending any large part of his time studying the satellite system or the use of FORTRAN Symbolic.

Reference 11 reports the work of Leach and Perrella in setting up the satellite system for general usage. By using their library tape in place of the regular computer center library tape the user has all the capabilities normally available with FORTRAN 60 plus the use of the satellite system and its subroutines. Briefly these take care of the complex tasks of setting up lines of communications between the various components of the system, packing and transmitting messages along these lines, and then breaking down the lines to resume normal computation.

Their routines are intended for general purpose users. As such, they sometimes lack features desirable for simulations. The simulation programmer may take advantage of this prior work by modifying certain of the subroutines to fit his simulation. This has been done in the case of Leach and Perrella's subroutine CHANGE. It was noted that this subroutine stopped a running program, gave control to the typewriter, processed messages from the typewriter, and upon signal, transferred

control back to the running program. All these features were desirable, but the specific messages that were processed were of little use in a simulation. CHANGE was modified to correct this and INCOMM resulted. The modification procedure will be described in detail later in this section.

It should be noted here that an on-line simulation does not necessarily imply operation from the satellite station. With the subroutine described in this section the simulation may also be executed and controlled from the console typewriter. In this mode the messages to the player and the inputs to the program from the player all go via the console typewriter. The satellite station is preferred for the example problem because of the superior display capability which is available. Other simulations that do not depend upon graphical information might find the console typewriter, especially in combination with the on-line printer, fully satisfactory.

The graphical presentation provided by Leach and Perrella may also be tailored to simulation needs. This can be done by modifying their subroutine SATGRAF. Since the means of accomplishing this parallels the method used to develop INCOMM, it will not be discussed here.

A large part of the work involved in writing an on-line simulation is identical to that involved for off-line simulations. All of this type of programming may be done using FORTRAN 60. The feature which distinguishes an on-line simulation from an off-line simulation is the

existence of a dialogue between the player and the running program. Unfortunately FORTRAN 60 is not adequate to create this dialogue. It is therefore essential to write a FORTRAN Symbolic subroutine to allow the player to interact with the computer and alter program parameters. Much of the framework for such a subroutine has already been written. The only task remaining for the individual user is to prepare sufficient coding to identify the typewriter messages he desires the players to enter. After he has written this small (30 lines in the sample program) amount of coding he may sandwich it into the existing framework which provides for storing and restoring index registers and monitor addresses, packing and transmitting the player's typewriter messages, and returning acknowledgements from the program when parameters have been successfully altered.

After the subroutine has been prepared it may be called by the FORTRAN 60 program whenever the programmer desires to give the player an opportunity to alter the progress of the game. When called, the subroutine delivers a typewriter message to the player then awaits an output from his keyboard. When the player has typed a complete message--which he signifies by pressing the "output" button on the DD-65 keyboard or the carriage return on the console typewriter--the subroutine examines the contents of the message. If "end" was typed the subroutine responds by typing "end" and exits to the main program. If any message other than "end" was typed the subroutine jumps to the portion of coding provided by the simulation programmer. His portion must systematically check individual characters within the message to determine its contents.

When unambigious identification is complete the subroutine must set variables held in common with the main program to appropriate values and then jump to a portion of the subroutine framework which delivers a response to the player and awaits his next typewriter message. An example will be followed through to clarify the details.

Subroutine MACHINE INCOMM accomplishes all of the above tasks. (See page 120.) Each time INCOMM is called it is possible to change any or all of the variables KEYCRS, KEYFIRE, or KEYSPD. As described above it is entered by the statement CALL INCOMM in the main program and it exits and returns control to the main program when the player types and outputs "end".

As the listing shows INCOMM can be divided into three sections. Block #1 and block #3 make up the framework described above. They may be used verbatum. If this is done the LOC statement must be used as is, and a HOL statement with END, ERROR, and ORDERS must be used. Block #2 applies to the specific simulation and must be provided by the simulation programmer. Since the other two entries in the HOL statement are used only in block #2 they may be replaced by any desired responses to the player. There is no limit to the number of these that may be employed, but none of them may exceed eight characters.

It may be noted that block #2 is entered from line 15 of block #1 after it has determined that "end" was not typed. The player originated typewriter message exists in a read buffer of 120 cells. Each cell contains only one BCD character. For example, if the player typed "C/S TO 1/3" the first cell of the read buffer would contain the BCD representation

of "C" or 63 octal. This first cell of the buffer is addressed in the subroutine as R. Then cell R+1 contains the BCD representation of "/" and cell R+2 contains "S". Similarly, each character of the message resides in consecutive single cells including an octal 20 for each space that was typed.

For the program to determine if an S was typed as the third character the following sequence is used. Enter the A register with the BCD representation of the letter S (22 octal). Check to see if the quantity in cell R+2 is this number. If it is not, go to some other labeled set of instructions to check for a different possible message (e.g., 1FIRE). If it is an "S", check the next instruction in the sequence. The group of instructions which does precisely this is:

```
ENI(0)              ENA(22B)          .
EQS(R+2)            SLJ(1FIRE)        .
(next instruction in sequence)        .
```

The pass instruction, ENI(0), is essential only to space the EQS into the upper half of an instruction. Other checks on the contents of the read buffer are made in a similar way. Once sufficient checks have been made to unambiguously identify the message one or more of the variables in common may be simply set as follows:

```
ENA (1)             STA(KEYSPD)       .
```

To inform the player that a parameter has been successfully set a message is returned to him. In the example problem the same message is used for several different parameters, so it is labeled (1RSPD) and

appears only once in block #2. The instruction SLJ(1RSPD) is used to jump to this labeled location. The simulation programmer may note that control jumps from the line labeled 1RSPD to a line within block #3 labeled 1OUT. This is the portion of coding which properly packs this message and sends it to the player via the DD-65 screen or the console typewriter. It also causes the program to await the next typewriter message from the player.

If an equality search is used which has only one expected outcome, the unexpected outcomes may be processed as errors. This may be handled as follows:

```
EQS(R+2)          SLJ(1ERR)
(next instruction for expected case)
```

If the equality is not satisfied, this will output a message, "ERROR" to the player and wait for him to output a new message.

Many other variations are possible and will suggest themselves in the context of a particular simulation. The above instructions are only intended to provide enough foundation to get a new user of the on-line simulation started with a running program. It does not indicate any degree of optimum utilization of the system.

Some readers may question why blocks #1 and #3 have not been coded into a separate subroutine so a user need not re-code them. This is due to the audience for which they are intended. It is expected that this may be the only assembly language subroutine they are forced to write. Considering this, the approach taken requires the least possible knowledge of symbolic coding and offers the smallest possible chance for errors.

APPENDIX IV

ON-LINE SIMULATION PROGRAM

The flow charts and program contained in this appendix refer to an on-line simulation played on the area shown below.

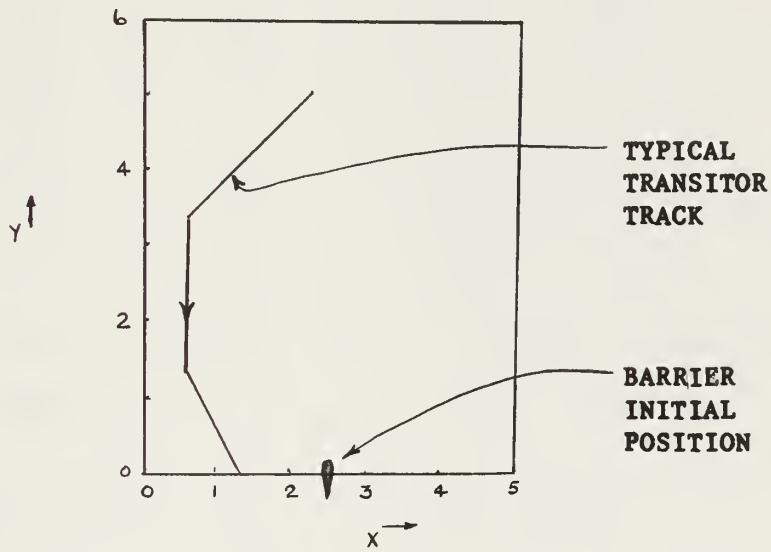
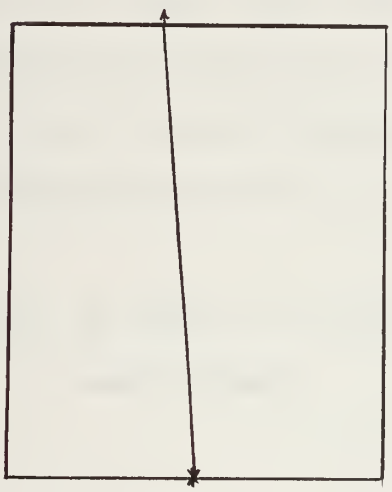


FIGURE 5

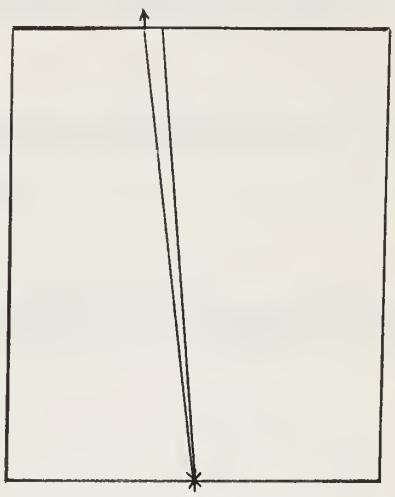
On-line simulation playing area

In this simulation the barrier submarine is free to move back and forth along the base line at the control of the player. The transitor is started from a random position along the line Y = 5. His speed and motion as well as his type of ship are all chosen randomly, but the player may override any of these if he desires. The player is presented bearings only information plotted on the above grid which appears on the

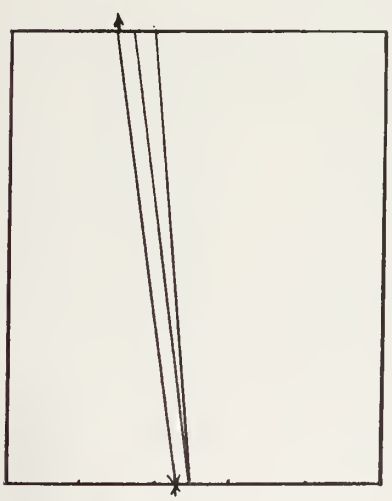
left tube of the DD-65. Four simulated steps of time after the second of which the barrier began to move to the left, would appear to the player as shown in Figure 6.



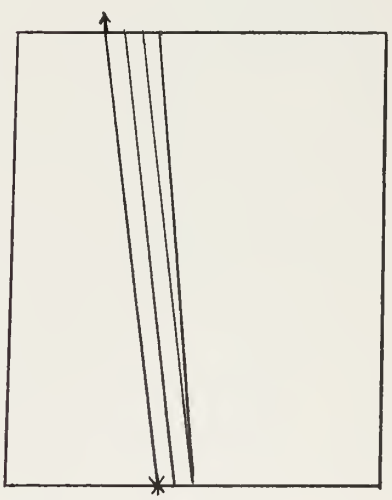
TIME 1



TIME 2



TIME 3



TIME 4

FIGURE 6

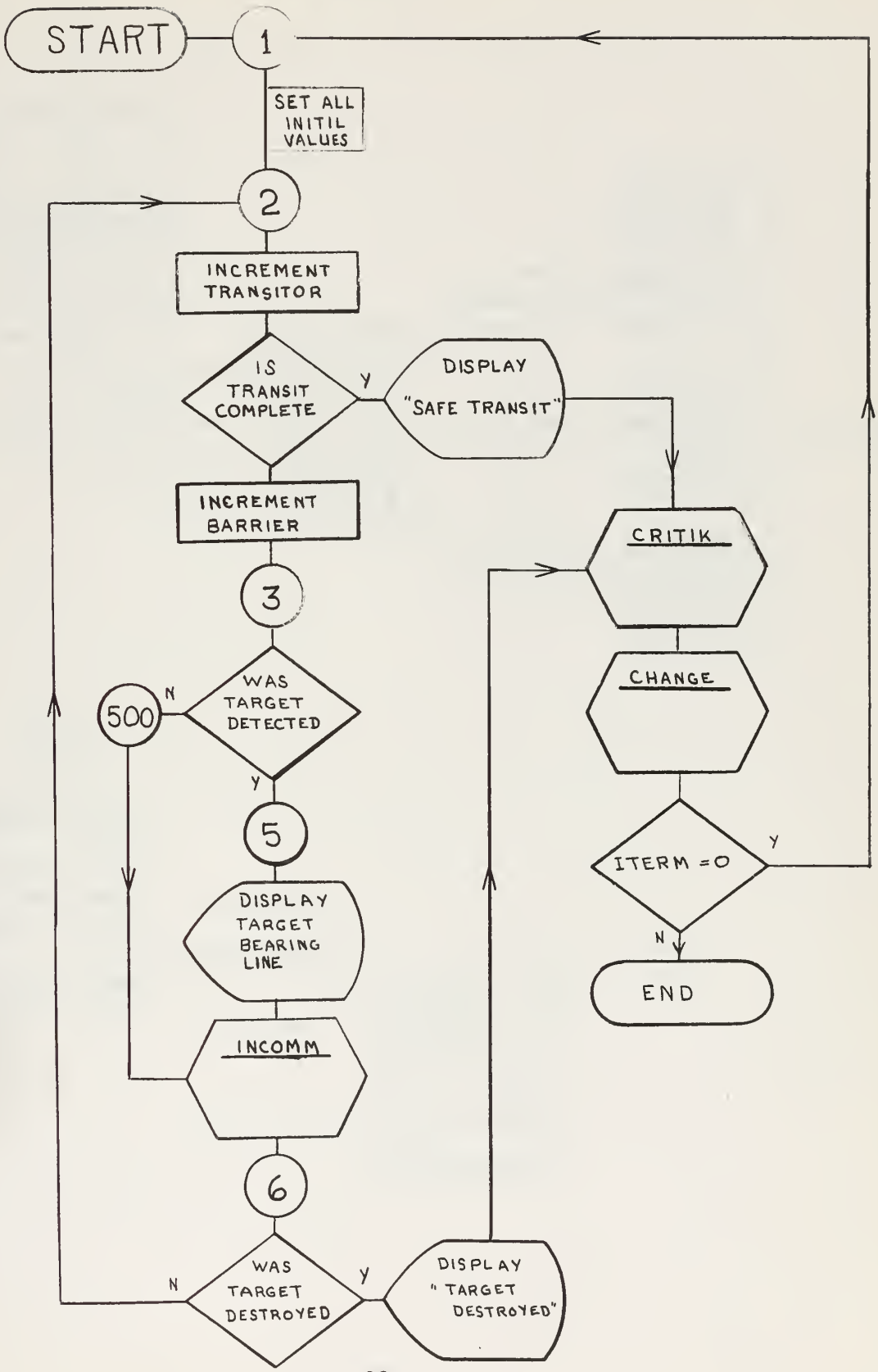
DD-65 scope presentation to player

By observing only the bearing information the player tries to maneuver his submarine to intercept and sink the transitor. His success or failure is evaluated by the computer; and, after the run has been completed, this evaluation, the previously unknown target data, the actual target track and the barrier submarines track are presented for the player's viewing.

The flow charts and the program listing which follows will serve to explain the details of the simulation.

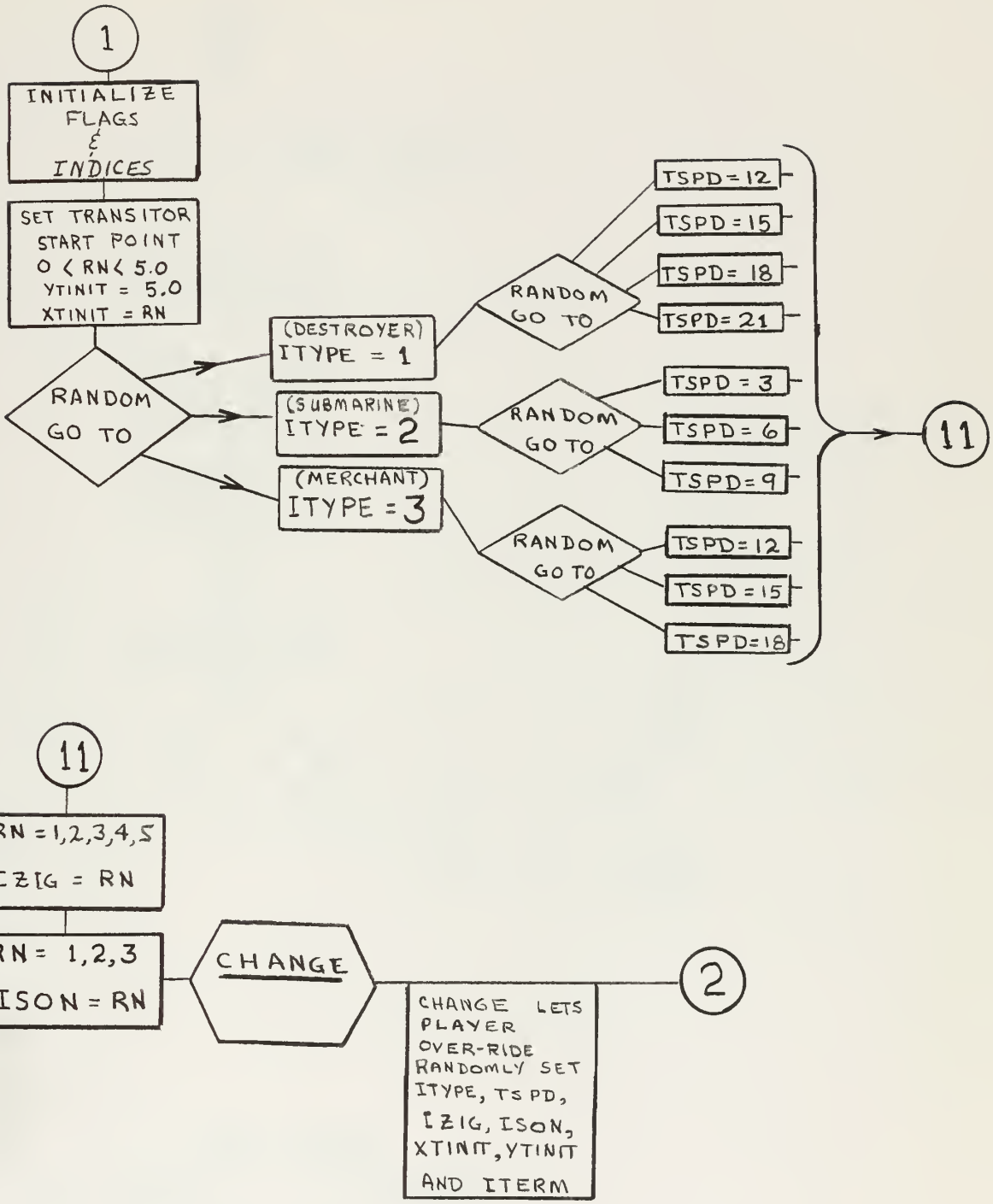
Note: RN refers to a random number generated by the computer using function RANF.

ON-LINE PROGRAM (SKELETON)



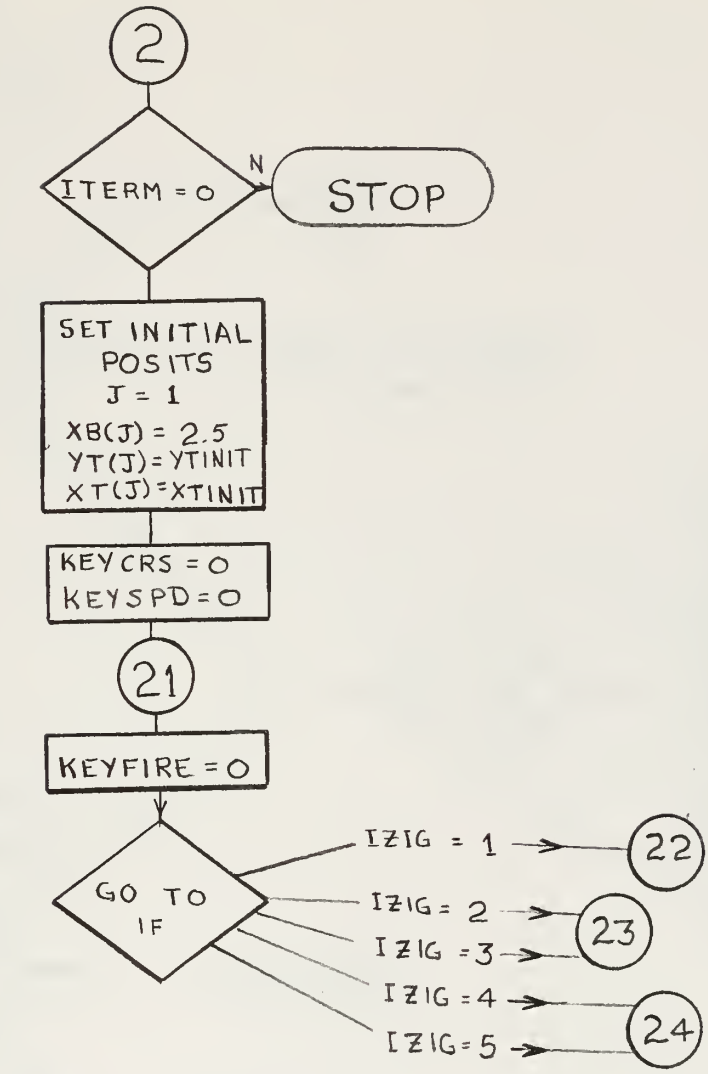
ON-LINE MAIN PROGRAM

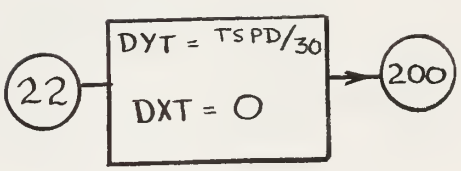
CHART 1



ON-LINE MAIN PROGRAM

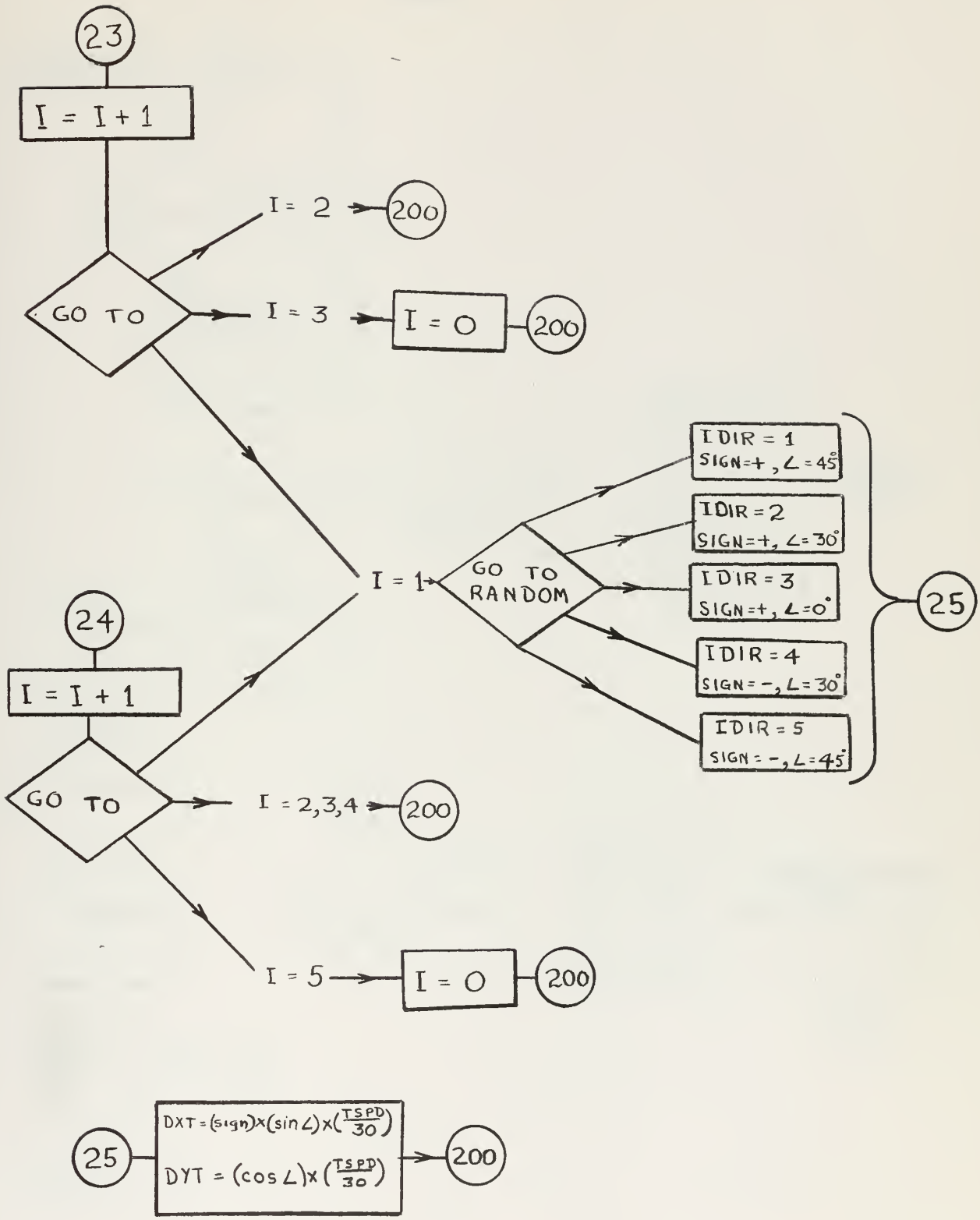
CHART 2





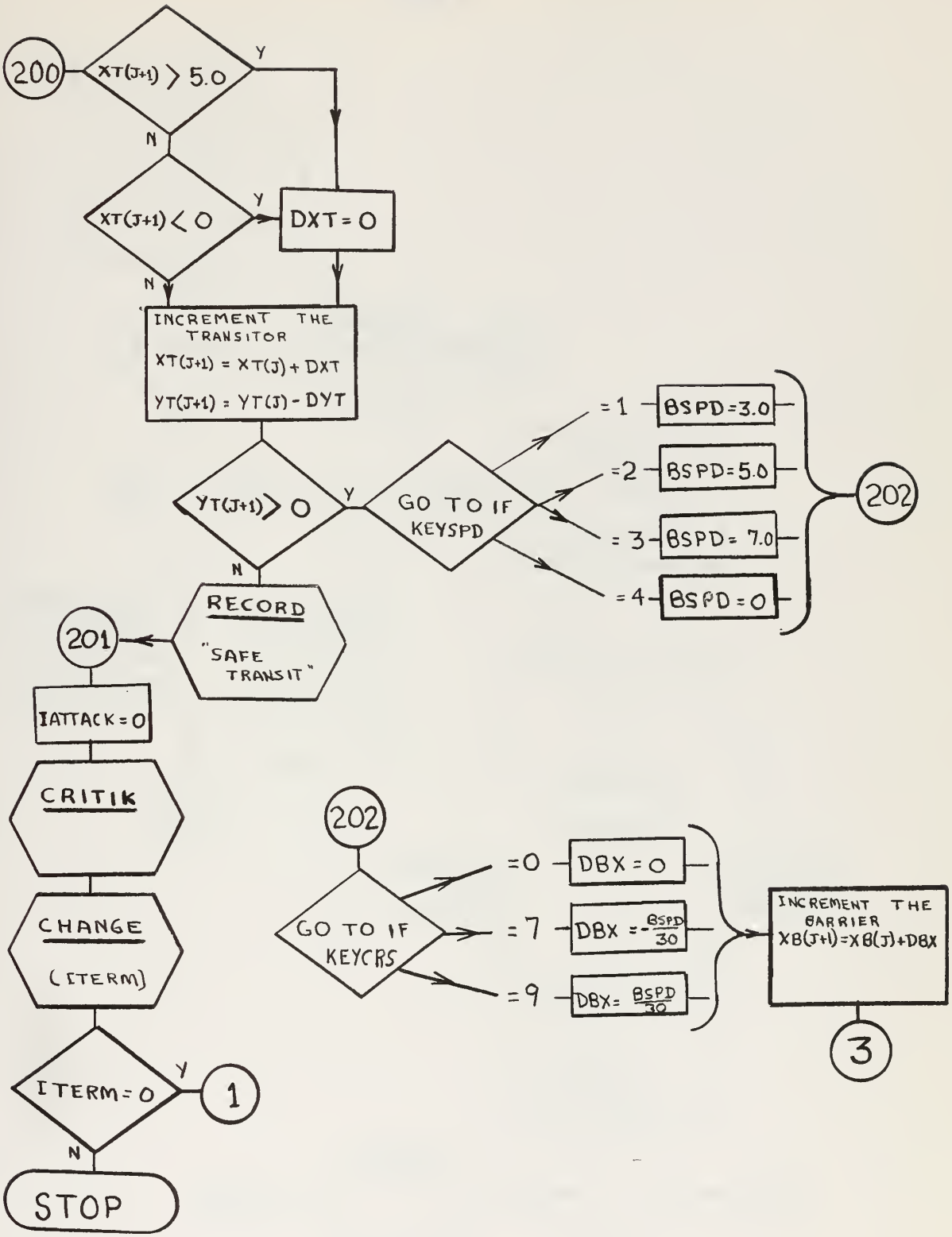
ON-LINE MAIN PROGRAM

CHART 3



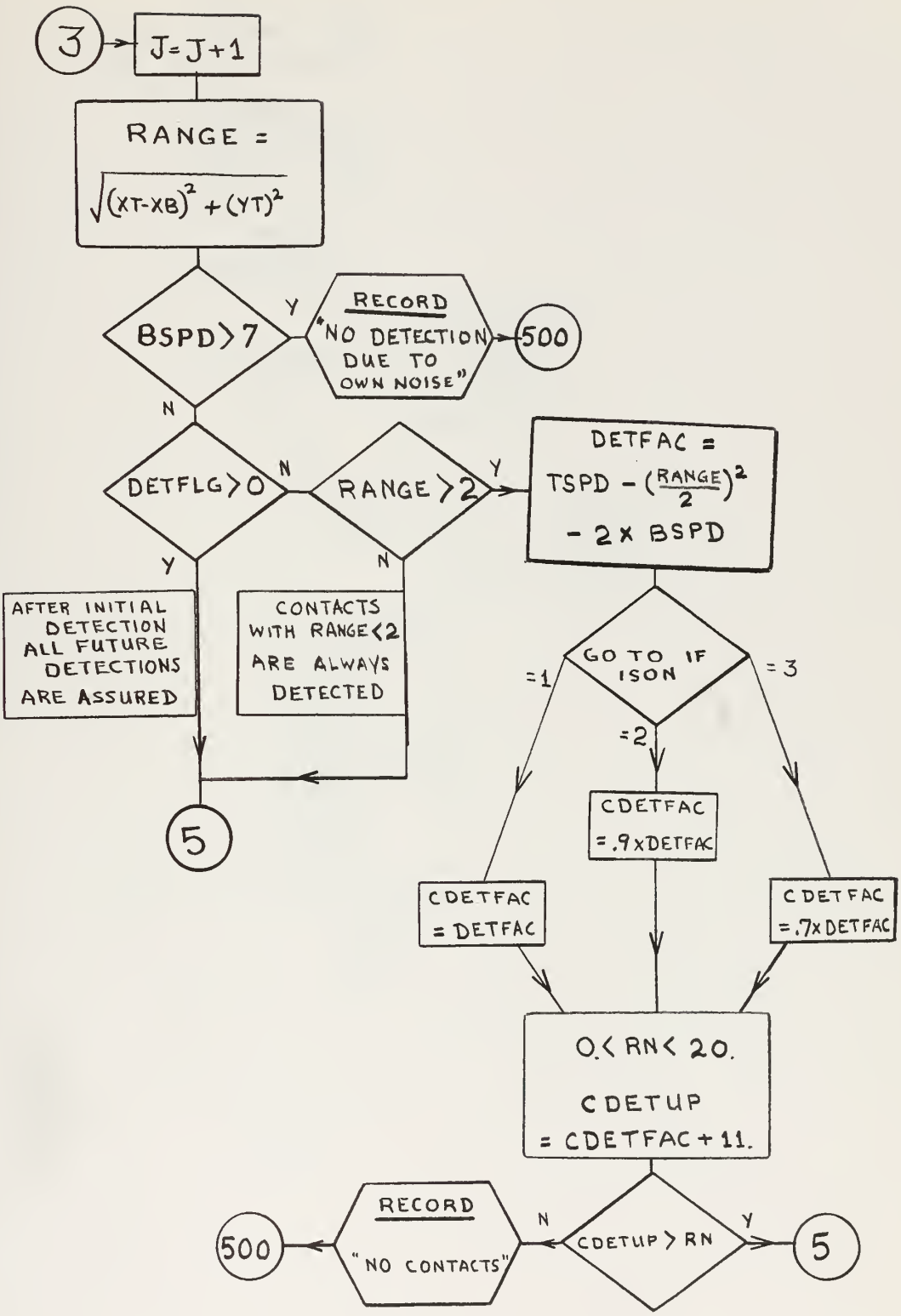
ON-LINE MAIN PROGRAM

CHART 4



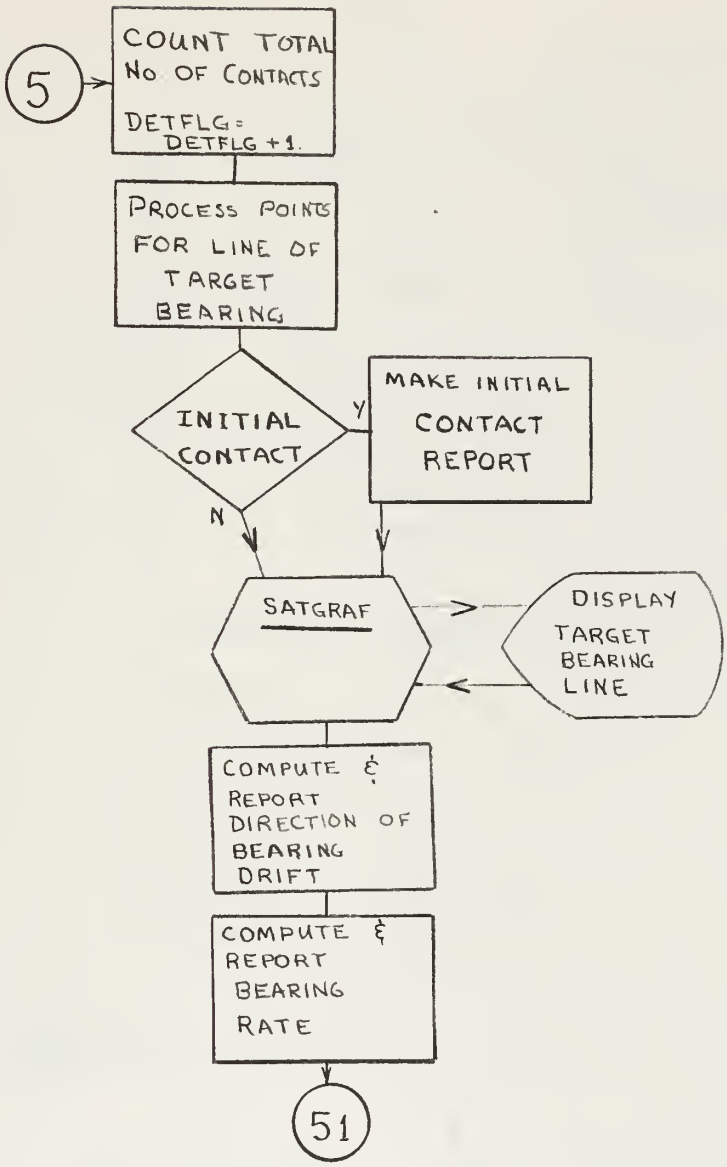
ON-LINE MAIN PROGRAM

CHART 5



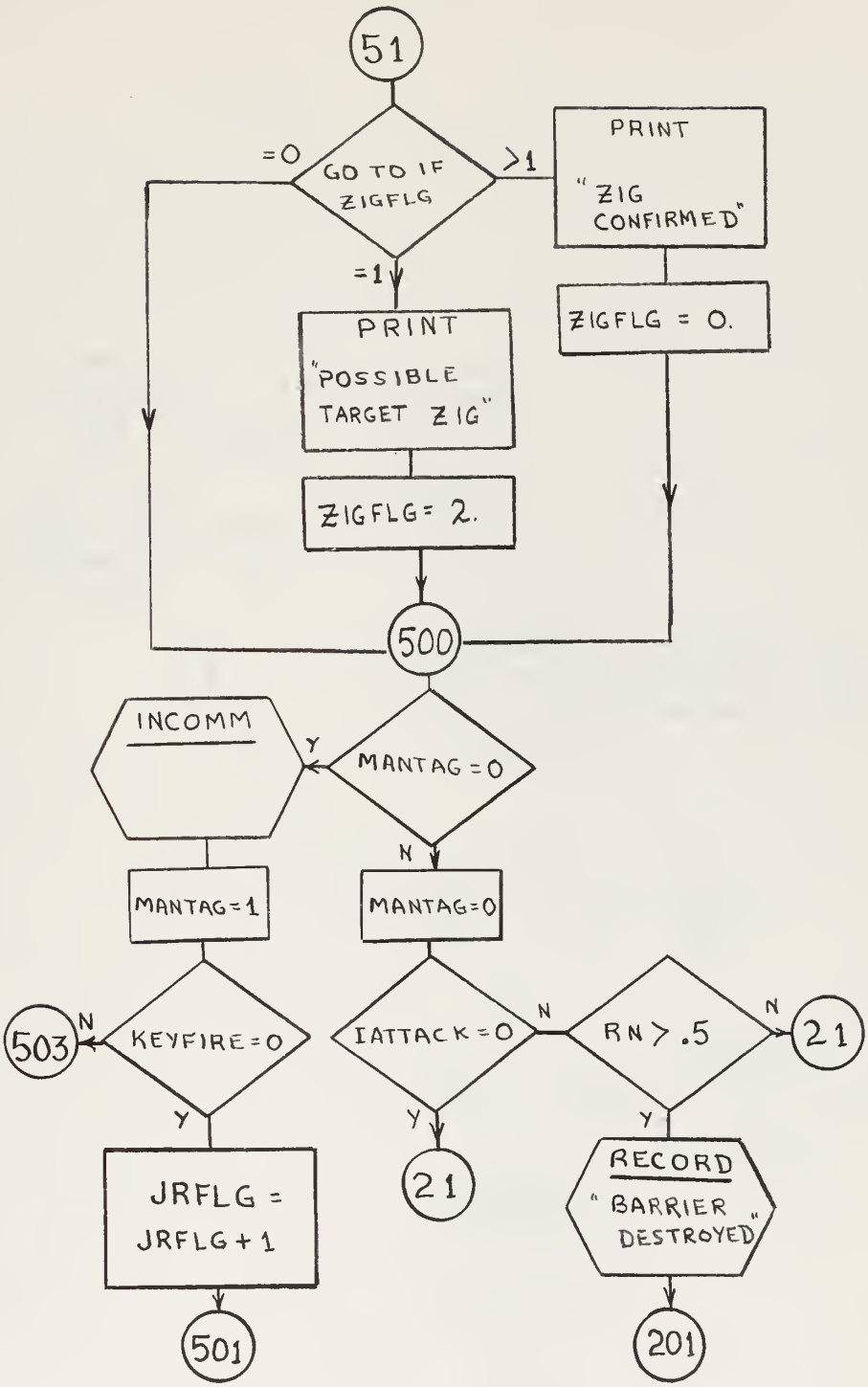
ON-LINE MAIN PROGRAM

CHART 6



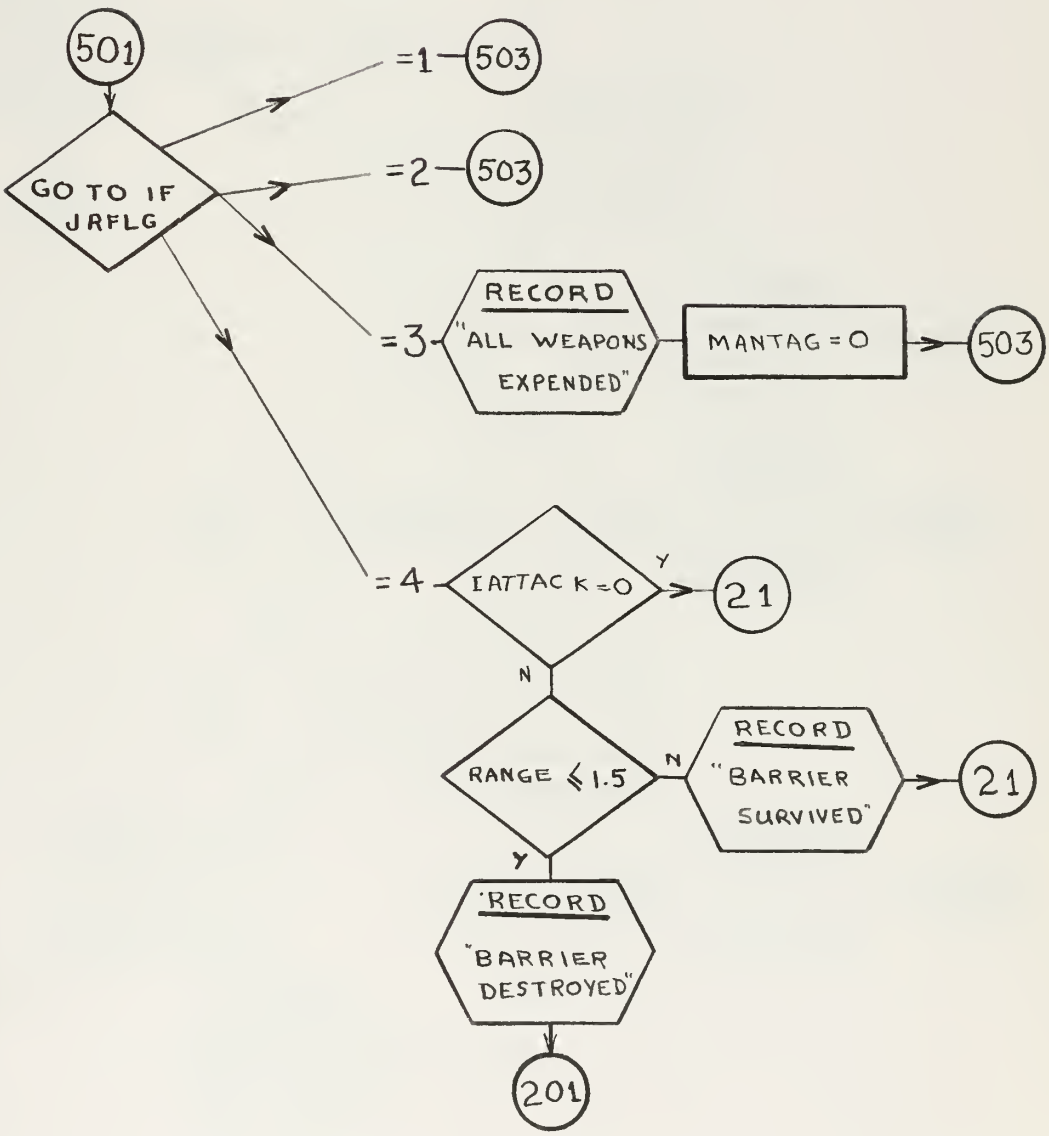
ON-LINE MAIN PROGRAM

CHART 7



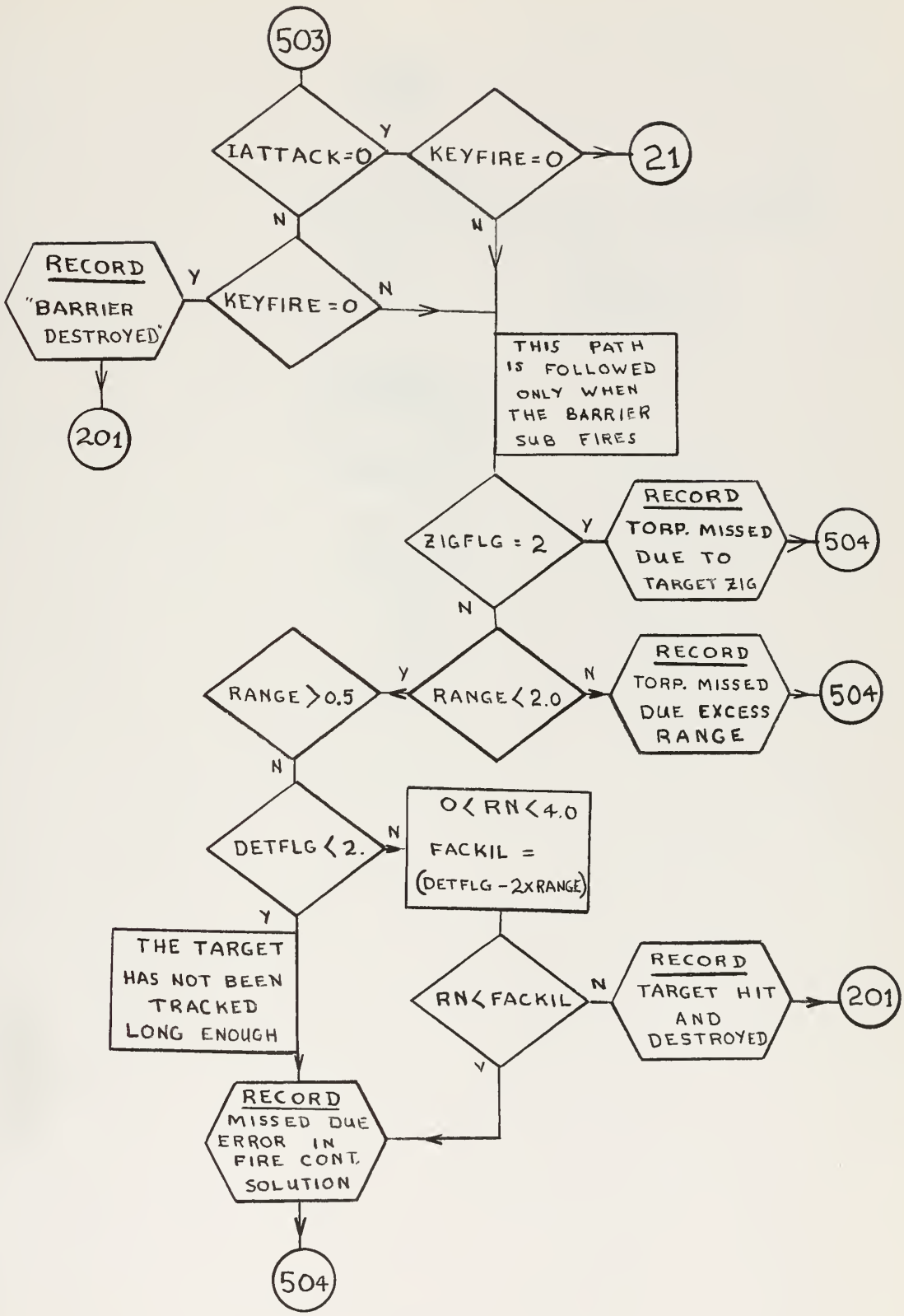
ON-LINE MAIN PROGRAM

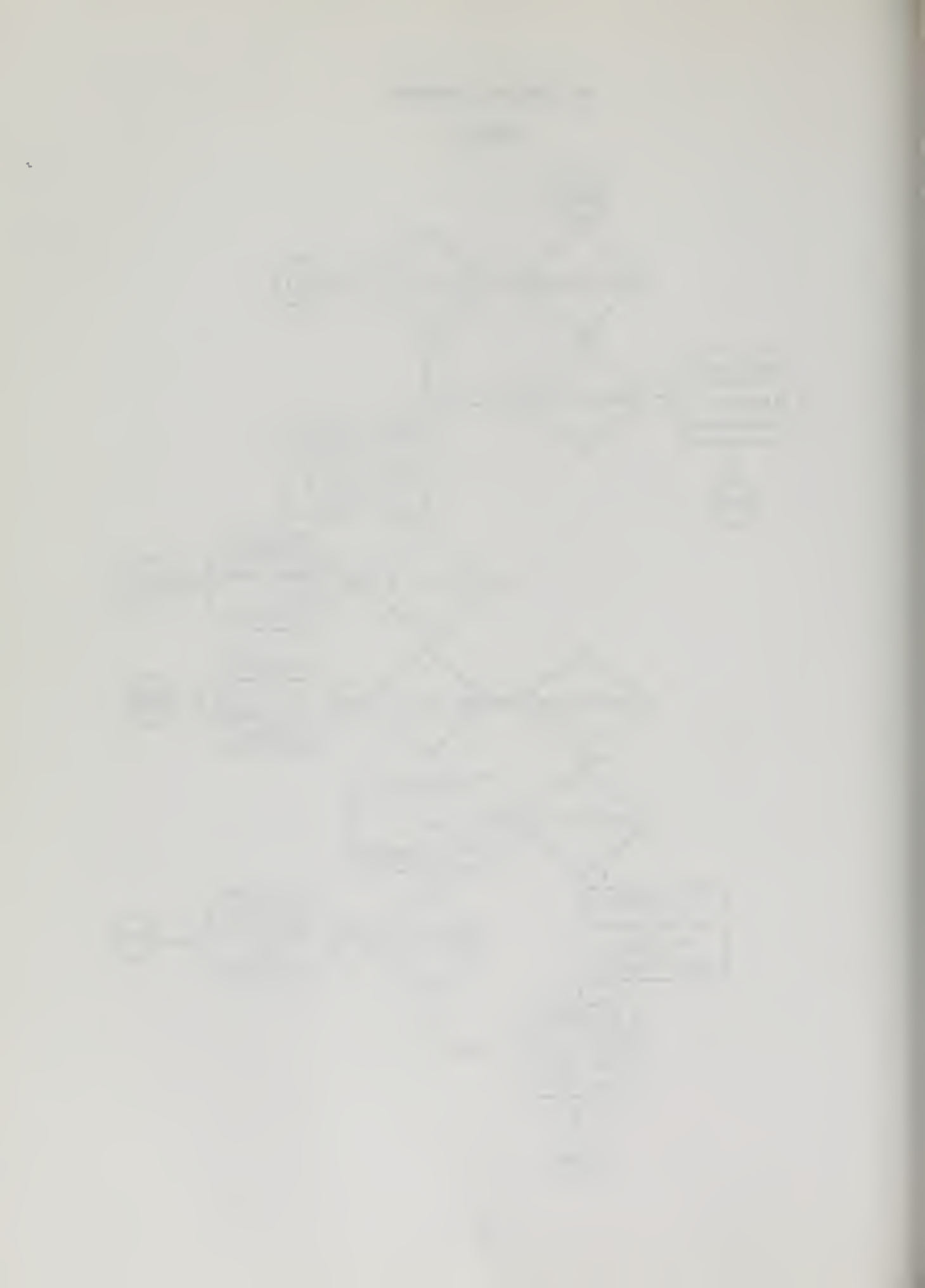
CHART 8



ON-LINE MAIN PROGRAM

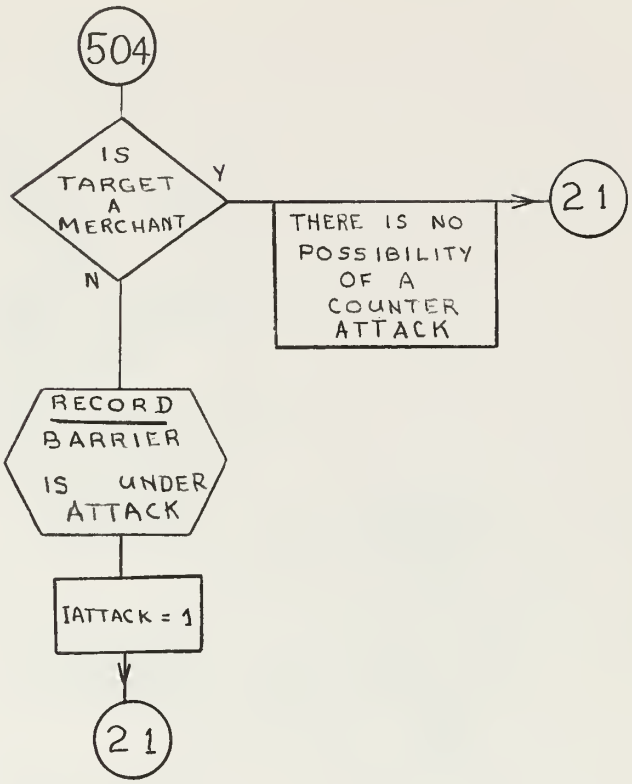
CHART 9

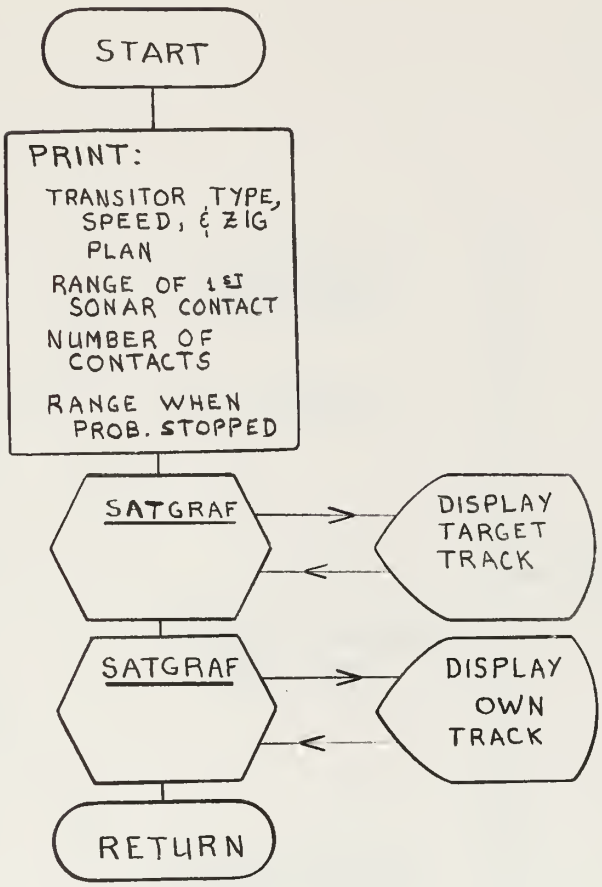


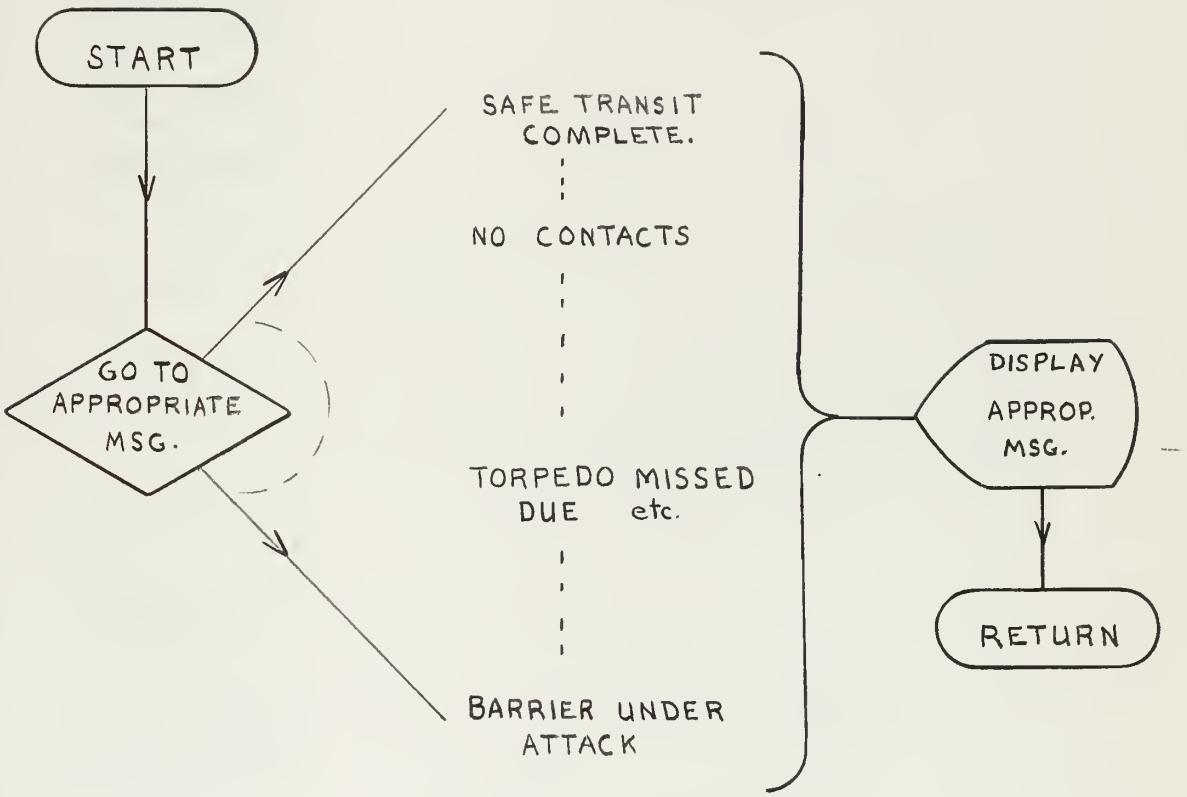


ON-LINE MAIN PROGRAM

CHART 10

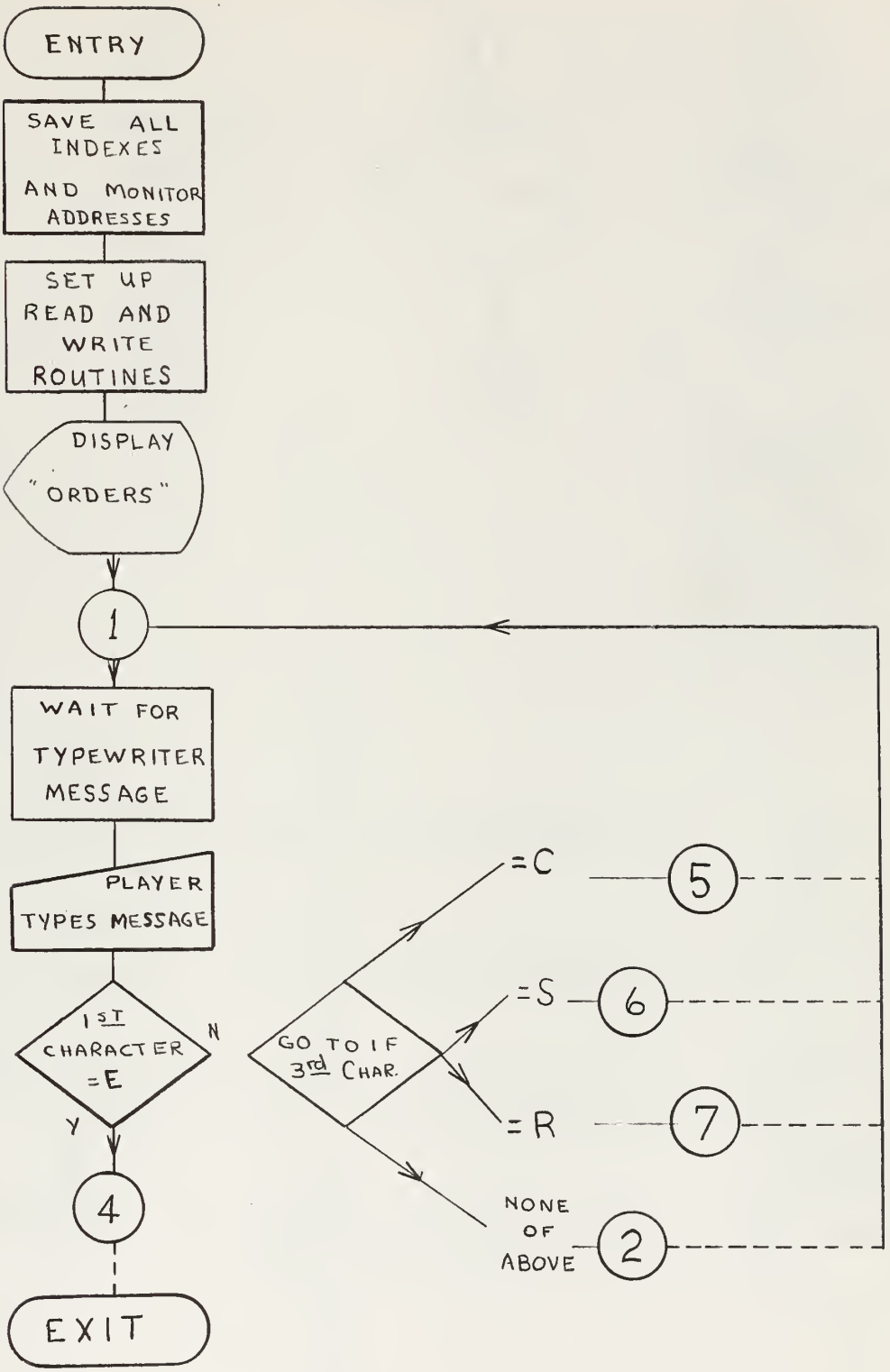



START
PRINT:
TRANSITOR TYPE, SPEED, & ZIG PLAN
RANGE OF 1ST SONAR CONTACT
NUMBER OF CONTACTS
RANGE WHEN PROB. STOPPED
SATGRAF
DISPLAY TARGET TRACK
SATGRAF
DISPLAY OWN TRACK
RETURN


START
GO TO APPROPRIATE MSG.
SAFE TRANSIT COMPLETE.
NO CONTACTS
TORPEDO MISSED DUE etc.
BARRIER UNDER ATTACK
DISPLAY APPROP. MSG.
RETURN

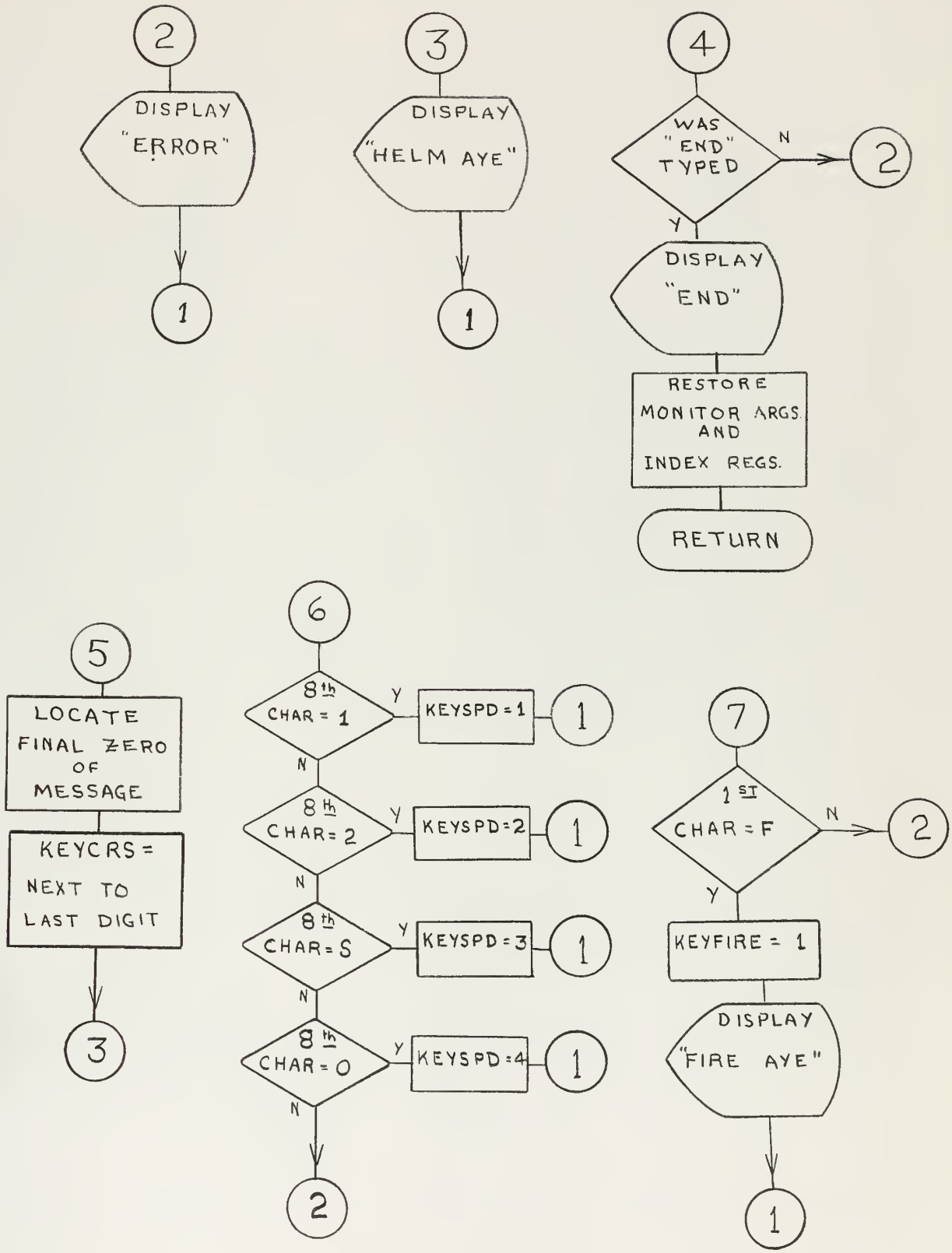
ON-LINE SUBROUTINE INCOMM

CHART 1



ON-LINE SUBROUTINE INCOMM

CHART 2



```
      PROGRAM SUB BAR 3
C        IN RUNNING THIS PROGRAM EACH GRAPH WILL CAUSE A 30 SECOND
C        DELAY. TO ELIMINATE THIS DELAY PRESS THE MASTER COMM FLAG
C        BUTTON. IT SHOULD REMAIN LIGHTED.  NOW PRESSING THE COMM
C        FLAG SET BUTTON AFTER EACH GRAPH IS DISPLAYED WILL ALLOW
C        THE PROGRAM TO PROCEED WITHOUT A DELAY.
C        THE PLOTS ON THE LEFT TUBE WILL REACH A SATURATION OF THE DD-
C        65 MEMORY.  WHEN THIS OCCURS THE MESSAGE DD 65 MEMORY FULL
C        WILL BE PRINTED.  WITHIN 30 SECONDS YOU MUST PRESS THE COMM
C        FLAG KEY. THIS CLEARS PREVIOUS POINTS AND PLOTS THE REMAINING
C        POINTS.
*  THE VARIABLES HAVE MEANINGS AS FOLLOWS
*  KEYCRS =0      COURSE =000    .KEYSPD = 1    SPEED =1/3
*  KEYCRS =9      COURSE =090    .KEYSPD = 2    SPEED =2/3
*  KEYCRS =7      COURSE 270     .KEYSPD = 3    SPEED =STD
*                                .KEYSPD = 4    SPEED =0
*                   KEYFIRE =0   NO ATTACK
*                   KEYFIRE =1   ATTACK ORDERED
C    ITYPE 1 =DESTROYER
C    ITYPE 2 =SUBMARINE
C    ITYPE 3 =MERCHANT
C        TARGETS MAY OR MAY NOT ZIG AS FOLLOWS
C              NO ZIG    IZIG = 1
C              SHORT LEG IZIG = 2 OR 3
C              LONG LEG  IZIG = 4 OR 5
C        SONAR CONDITIONS MAY ALTER MAXIMUM SONAR RANGES AS FOLLOWS
C              ISON =  1         EXCELLENT SONAR CONDITIONS
C              ISON =  2     .9* EXCELLENT SONAR RANGES
C              ISON =  3     .7* EXCELLENT SONAR RANGES
C        TARGETS AND BARRIER ARE MOVED IN STEPS SIMULATING 1 MINUTE OF
C        REAL TIME. AN OPPORTUNITY TO MANUEVER THE BARRIER IS PROVIDED
C        AFTER EACH OF THESE MOVES. THIS OPPORTUNITY IS SIGNIFIED BY
C        A TYPED MESSAGE -ORDERS-
*        THE ONLY POSSIBLE RESPONSES ARE
*        C/C TO 000.                    C/S TO 1/3.
*        C/C TO 090.                    C/S TO 2/3.
*        C/C TO 270.                    C/S TO STD.
```

```
*        C/S TO 0.                              FIRE.
*        ANY VARIATION OF THE ABOVE WILL CAUSE THE DD-65 TO PRINT ERROR.
*        WHEN YOU HAVE SUCCESSFULLY ENTERED THE DESIRED COMMANDS,OR IF
*        YOU DESIRE TO ENTER NONE,TYPE END.
      DIMENSION XT(100),YT(100),XB(100)
      DIMENSION X(54),Y(54)
      COMMON XT,YT,XB,INAME1,INAME2, DETFLG, RANGE, RANGEIN,
     1TSPD, IZIG, J, KEYCRS, KEYSPD, KEYFIRE,KARROW,STPOINT,FINLPT
      COMMON NOISE
      IR=7495238
    1 I=0
      J=0
      JFRFLG =0
      KARROW =0
      ZIGFLG=0.0
      IATTACK = 0
      ITERM=0
      SC=30.
      DETFLG=0.
      PRINT 7900
      PRINT 7901
 7900 FORMAT(//49HINSTRUCTIONS-  EACH TIME THE WORD -ORDERS- OR THE   /
     1              15X        32HWORD -BEGIN- IS PRINTED YOU MUST  /
     2              15X        34HRESPOND.  IN EVERY CASE TYPING END  /)
 7901 FORMAT(       15X        34HAND PUSHING THE OUTPUT BUTTON WILL /
     1              15X        26HALLOW THE GAME TO PROCEED. )
C        DETERMINE START POINT FOR TRANSITOR
      RAND=RANF(IR)
      XTINIT=5.*RAND
      YTINIT=5.
C        DETERMINE TYPE TRANSITOR
      TYPE = RANF(IR)
      TYPEUP=3.*TYPE+1.0
      ITYPE=TYPEUP
C        DETERMINE TRANSITOR SPEED
      RAND=RANF(IR)
      RANDUP=RAND*5.+1.0
```

```
      ISPD=RANDUP
      GO TO (10,20,30),ITYPE
   10 GO TO (11,12,12,13,14),ISPD
   11 TSPD=12.
      GO TO 60
   12 TSPD=15.
      GO TO 60
   13 TSPD=18
      GO TO 60
   14 TSPD=21.
      GO TO 60
   20 GO TO (21,22,22,23,12),ISPD
   21 TSPD=3.
      GO TO 60
   22 TSPD=6.
      GO TO 60
   23 TSPD=9.
      GO TO 60
   30 GO TO (11,12,12,13,13),ISPD
   60 ZIG=RANF(IR)
      ZIGUP=5.*ZIG+1.0
      IZIG=ZIGUP
      SON=RANF(IR)
      SONUP=3.*SON+1.0
      ISON=SONUP
      PRINT 8000
      PRINT 8001
      PRINT 8002
      PRINT 8011
      PRINT 8003
 8000 FORMAT(//42HVARIABLES ARE ITYPE,TSPD,IZIG,ISON,XTINIT,)
 8001 FORMAT(14X,12HYTINIT,ITERM)
 8002 FORMAT  (42HTO ALTER THESE VARIABLES USE THE PROCEDURE  /
     1          21HOF SUBROUTINE CHANGE.  )
 8003 FORMAT ( 26HTO START THE RUN TYPE END.  )
C        SETTING ITERM TO ANY INTEGER OTHER THAN ZERO WILL
C        TERMINATE THE GAME
```

```
      CALL CHANGE(ITYPE,TSPD,IZIG,ISON,XTINIT,YTINIT,ITERM)
      IF(ITERM) 80,81,80
   80 STOP
   81 CONTINUE
      GO TO (91,92,93),ITYPE
   91 INAME1=8HDESTROYE
      INAME2=1HR
      GO TO 94
   92 INAME1=8HSUBMARIN
      INAME2=1HE
      GO TO 94
   93 INAME1=8HMERCHANT
      INAME2=1H
C        BARRIER SUBMARINE INITIAL
C        STATION AND STATUS
   94 XBINIT=2.5
      YB=0
      KEYCRS=0
      KEYSPD=4
      J=1
      XT(J)=XTINIT
      YT(J)=YTINIT
      XB(J)=XBINIT
C        ITERATIVE PART OF GAME COMMENCES
   95 KEYFIRE=0
      GO TO (96,97,97,100,100),IZIG
   96 DYT=TSPD/SC
      DXT=0.
      GO TO 175
   97 I=I+1
      IF(I-2)105,175,98
   98 I=0
      GO TO 175
  100 I=I+1
      GO TO (105,175,175,175,175,98),I
C        SELECT DIRECTION OF TARGET MOTION
  105 DIR=RANF(IR)
```

```
      DIRUP=5.*DIR+1.0
      IDIR=DIRUP
      ZIGFLG=1.0
  106 GO TO (110,120,96,140,150),IDIR
  110 DXT=(TSPD/SC)*.707
      DYT=DXT
      GO TO 175
  120 DXT=(TSPD/SC)*.5
      DYT=.86*(TSPD/SC)
      GO TO 175
  140 DXT=-(TSPD/SC)*.707
      DYT=(TSPD/SC)*.707
      GO TO 175
  150 DXT=-(TSPD/SC)*.5
      DYT=.86*(TSPD/SC)
C        CHECK TO INSURE TARGET DOES NOT
C        REMAIN OUTSIDE LANE BOUNDARY
  175 IF(XT(J)+DXT-5.0)180,180,177
  177 DXT=0
      GO TO 190
  180 IF(XT(J)+DXT)177,190,190
C        INCREMENT TRANSITOR
  190 XT(J+1)=XT(J)+DXT
      YT(J+1)=YT(J)-DYT
C        CHECK FOR COMPLETE TRANSIT
      IF(YT(J+1))200,200,300
  200 CALL RECORD (1)
  201 IATTACK=0
      CALL CRITIK
      PRINT 8010
 8010 FORMAT(//18HVARIABLE  IS ITERM)
      PRINT 8011
 8011 FORMAT(32HSET ITERM TO 1 TO TERMINATE GAME)
      PRINT 8012
 8012 FORMAT(30HTO SET UP ANOTHER RUN TYPE END)
      CALL CHANGE (ITERM)
      IF(ITERM) 80,1,80
```

```
C          DETERMINE BARRIER SPEED
  300 GO TO (310,320,330, 340),KEYSPD
  310 BSPD=3.
      GO TO 350
  320 BSPD=5.
      GO TO 350
  330 BSPD=7.
      GO TO 350
  340 BSPD=0.
      GO TO 355
C          DETERMINE DIRECTION OF BARRIER MOTION
  350 IF(KEYCRS - 7)  355, 360, 365
  355 DBX =0.
      GO TO 370
  360 DBX=-BSPD/SC
      GO TO 370
  365 DBX= BSPD/SC
  370 XB(J+1)=XB(J)+ DBX
      J=J+1
C          DETERMINE RANGE
C          AND DETECTION FACTOR
     RANGE=SQRTF((XT(J)-XB(J))**2 +YT(J)**2)
      IF (BSPD-7.)380,375, 375
  375 CALL RECORD (2)
      NOISE = 1
      CALL SATGRAF (1,54,X,Y)
      NOISE = 0
      GO TO  500
  380 IF(DETFLG) 382, 382, 400
C          WAS THERE A PREVIOUS DETECTION. IF SO DETECTION ASSURED
C          IF RANGE IS LESS THAN 2.0 DETECTION IS ALSO ASSURED
  382 IF(RANGE-2.)  400, 400, 383
  383 DETFAC=(  TSPD -(RANGE/2.)**2 - BSPD*2.)
*          CORRECT FOR SONAR CONDITIONS
      GO TO (384,385,386), ISON
  384 CDETFAC=DETFAC
      GO TO 390
```

```
  385 CDETFAC=.9*DETFAC
      GO TO 390
  386 CDETFAC=.7*DETFAC
  390 RDEC=RANF(IR)
      RDECUP=RDEC* 20.
      CDETUP=CDETFAC + 11.
      IF (CDETUP-RDECUP)395,398,398
  395 CALL RECORD (3)
      GO TO 500
C        REPORT AN INITIAL CONTACT
  398 CALL RECORD (15)
      GO TO 405
  400 IF(RANGE -1.0) 401,401,404
C        MAKE SPECIAL REPORT FOR VERY CLOSE CONTACT
  401 CALL RECORD (11)
      GO TO 405
 404  CALL RECORD(4)
  405 DETFLG =DETFLG+1.0
      SLOPE=(XT(J)-XB(J))/YT(J)
C        PROCESS POINTS FOR BEARING PLOT
      K=1
      Y(K)=0
      X(K)=XB(J)
      STPOINT=X(1)
      DO 410 K=1,54
      Y(K+1)=Y(K)+.12
      X(K+1)=X(K)+SLOPE*.12
      IF(X(K+1)+1.0) 407,407,408
  407 X(K+1)=-1.0
      GO TO 410
  408 IF(X(K+1)-5.) 410,409,409
  409 X(K+1)=5.0
  410 CONTINUE
      FINLPT=X(51)
      IF(DETFLG-1.0) 420, 420, 430
C        PROCESS INITIAL CONTACTS
  420 BEARING = ATANF(SLOPE) * 57.296
```

```
      IF (BEARING) 421,426,426
  421 BEARING = 360. + BEARING
  425 PRINT 4250, BEARING
      GO TO 429
  426 IF(BEARING-10.)427,428,428
  427 PRINT 4270, BEARING
      GO TO  429
  428 PRINT 4280, BEARING
 4250 FORMAT(31HSONAR REPORTS NOISE LEVEL BEARS ,F5.0)
 4270 FORMAT (36HSONAR REPORTS NOISE LEVEL BEARING 00, F2.0)
 4280 FORMAT (35HSONAR REPORTS NOISE LEVEL BEARING 0, F3.0)
  429 KARROW = 1
      CALL SATGRAF(2, 54,X,Y,0,1,1.0,1.0)
      KARROW =0
      OLDSLOP=SLOPE
      RANGEIN = RANGE
      GO TO  500
C        PROCESS OLD CONTACTS
  430 KARROW = 1
      CALL SATGRAF (1,54,X,Y)
      KARROW =0
      IF(OLDSLOP  - SLOPE) 440,450, 460
  440 CALL RECORD (5)
      GO TO 470
  450 CALL RECORD (6)
      GO TO 470
  460 CALL RECORD (7)
  470 CURBR=ATANF(SLOPE)*57.296
      PRVBR=ATANF(OLDSLOP)*57.296
C        COMPUTE THE BEARING RATE
      BRATE=PRVBR-CURBR
      IF(BRATE) 471,472,473
  471 BRATE = -1. * BRATE
      PRINT  4710, BRATE
      GO TO 475
  472 PRINT 4720
      GO TO 475
```

```
  473 PRINT 4730,BRATE
  475 OLDSLOP=SLOPE
      IF(ZIGFLG-1.0)500,477,478
  477 PRINT 4770
      ZIGFLG=2.0
      GO TO 500
  478 PRINT 4780
      ZIGFLG=0.0
 4710 FORMAT (22HBEARING RATE IS RIGHT, F5.2,17H DEGREES PER MIN.)
 4720 FORMAT (21HBEARING RATE IS ZERO.)
 4730 FORMAT (20HBEARING RATE IS LEFT,  F5.2 ,20H DEGREES PER MINUTE.)
 4770 FORMAT (20HPOSSIBLE TARGET ZIG.)
 4780 FORMAT (14HZIG CONFIRMED.)
      GO TO 500
  488 CALL INCOMM
C         INCOMM IS A SUBROUTINE TO ALLOW BARRIER MANEUVERS.
C         AFTER THE APPROACH OFFICER HAS HAD AN OPPORTUNITY
C         TO MANEUVER, THE PROGRAM CHECKS FOR AN ATTACK ORDER AND
C         TO SEE IF THE BARRIER IS UNDER ATTACK.  IF NEITHER OF
C         THESE ANOTHER ITERATION THROUGH THE PRIOR PART OF THE
C         PROGRAM COMMENCES.  OTHERWISE  THE PROGRAM CONTINUES.
  489 MANTAG = 1
C         COUNT NUMBER OF TIMES BARRIER FIRES
      IF(KEYFIRE) 490,501,490
  490 JFRFLG = JFRFLG + 1
      GO TO (501,501,550,560,560,560),JFRFLG
  500 IF(MANTAG) 498,488,498
  498 MANTAG = 0
      IF (IATTACK) 499,95,499
  499 DNGR = RANF(IR)
      IF (DNGR - .5) 95,95,545
  501 IF(IATTACK) 502,503,502
  502 IF(KEYFIRE) 504,545,504
  503  IF (KEYFIRE)504,95,504
  504 IF(ZIGFLG-2.0)510,505,510
  505 CALL RECORD(12)
      GO TO 530
```

```
  510 IF(RANGE - 2.0)  512, 514,513
  512 IF(RANGE -0.5)   520, 520, 514
  513 CALL RECORD (8)
      GO TO 530
  514 IF(DETFLG - 6.0)  515,516,516
  515 CALL RECORD (10)
      GO TO 530
  516 FACKIL = DETFLG - 2.* RANGE
C        THE ABOVE HIT PROBABILITY IS BASED ON
C        THE NUMBER OF CONTACTS ON THE TARGET PRIOR
C        TO FIRING AND ON THE RANGE
      RKILL = RANF(IR)
      RKILLUP=RKILL* 4.0
      IF(RKILLUP - FACKIL) 520, 520, 525
  520 CALL RECORD(9)
      GO TO 201
  525 CALL RECORD (10)
C        IF THE TARGET IS NOT A MERCHANT AN UNSUCCESSFUL ATTACK WILL
C        BETRAY THE SUBS PRESENCE AND PROVOKE A COUNTER ATTACK.
  530 IF(INAME1-8HMERCHANT) 540,95,540
  540 CALL RECORD (13)
      IATTACK=1
      GO TO 95
  545 CALL RECORD (14)
      GO TO 201
  550 CALL RECORD (16)
      MANTAG = 0
      GO TO 501
  560 MANTAG=0
      IF(IATTACK) 570,95,570
  570 IF(RANGE-1.5)545,545,575
  575 CALL RECORD (17)
      GO TO 95
      END
```

```
      SUBROUTINE CRITIK
      DIMENSION XT(100), YT(100), XB(100)
      DIMENSION YB(100)
      COMMON XT,YT,XB,INAME1,INAME2, DETFLG, RANGE, RANGEIN,
     1TSPD, IZIG, J, KEYCRS, KEYSPD, KEYFIRE,KARROW,STPOINT,FINLPT
      COMMON NOISE
      K=1
      YB(K) = 0.01
      DO 10 K=1,100
   10 YB(K+1) = YB(K) + .008
      PRINT 9002, INAME1, INAME2
      PRINT 9004, TSPD,IZIG
      PRINT 9006, RANGEIN
      PRINT 9010, DETFLG
      PRINT 9020, RANGE
      PRINT 9030
      CALL SATGRAF (2,  J, XT, YT, 0, 1, 1.0, 1.0)
      PRINT 9040
      CALL SATGRAF (1,J,XB,YB,  )
      RETURN
 9002 FORMAT(16HTARGET TYPE WAS ,A8, A1)
 9004 FORMAT(15HTARGET SPEED = , F5.1,17H ZIG PLAN IZIG = , I2, 2X,
     1          9HIN EFFECT )
 9006 FORMAT(33HTARGET FIRST CONTACTED AT RANGE =,  F4.1)
 9010 FORMAT(39HTOTAL NUMBER OF DETECTIONS ON TARGET = , F4.1 //)
 9020 FORMAT(39HRANGE TO TARGET WHEN PROBLEM STOPPED = , F4.1 //)
 9030 FORMAT(53HA PLOT OF TARGET TRACK WILL BE SHOWN ON THE LEFT TUBE)
 9040 FORMAT(35HOWN SHIP TRACK WILL NOW BE PLOTTED.)
      END
```

```
      SUBROUTINE RECORD(IREC)
      GO TO (951, 952, 953, 954, 955, 956, 957, 958, 959, 960,
     1       961,962,963,964,965,966,967),IREC
  951 PRINT 9510
      RETURN
  952 PRINT 9520
      RETURN
  953 PRINT 9530
      RETURN
  954 PRINT 9540
      RETURN
  955 PRINT 9550
      RETURN
  956 PRINT 9560
      RETURN
  957 PRINT 9570
      RETURN
  958 PRINT 9580
      RETURN
  959 PRINT 9590
      RETURN
  960 PRINT 9600
      RETURN
  961 PRINT 9610
      RETURN
  962 PRINT 9620
      RETURN
  963 PRINT 9630
      RETURN
  964 PRINT 9640
      RETURN
  965 PRINT 9650
      RETURN
  966 PRINT 9660
      RETURN
  967 PRINT 9670
      RETURN
```

```
9510 FORMAT(23HSAFE TRANSIT COMPLETED.)
9520 FORMAT(35HNO DETECTION DUE TO OWN SHIP NOISE.)
9530 FORMAT(26HSONAR REPORTS NO CONTACTS.)
9540 FORMAT(22HSONAR HOLDS A CONTACT.)
9550 FORMAT(37HSONAR REPORTS BEARINGS DRAWING RIGHT.)
9560 FORMAT(30HSONAR REPORTS BEARINGS STEADY.)
9570 FORMAT(36HSONAR REPORTS BEARINGS DRAWING LEFT.)
9580 FORMAT(35HTORPEDO MISSED DUE TO EXCESS RANGE.  /
    1       31HANOTHER ATTACK MAY BE POSSIBLE.)
9590 FORMAT(42HTARGET HIT AND DESTROYED. CONGRATULATIONS.)
9600 FORMAT(53HTORPEDO MISSED DUE TO ERROR IN FIRE CONTROL SOLUTION./
    1       22HNO CHANCE TO REATTACK.)
9610 FORMAT (42HSONAR REPORTS MACHINERY NOISE FROM TARGET.)
9620 FORMAT (49HTORPEDO MISSED DUE TO TARGET ZIG AT TIME OF FIRE. /
    1        31HANOTHER ATTACK MAY BE POSSIBLE.)
9630 FORMAT (34HBARRIER UNDER ATTACK BY TRANSITOR.)
9640 FORMAT (15HBARRIER DESTROYED.)
9650 FORMAT(///// 43HSONAR REPORTS A NOISE LEVEL FROM THE NORTH. )
9660 FORMAT(////38HYOUR FINAL WEAPONS HAVE BEEN LAUNCHED. /
    1           39HYOU ARE UNABLE TO MAKE FURTHER ATTACKS. )
9670 FORMAT(29HBARRIER SURVIVED LAST ATTACK.)
     END
```

```
      MACHINE INCOMM
      COMMON XT,YT,XB,INAME1,INAME2, DETFLG, RANGE, RANGEIN,
     1TSPD, IZIG, J, KEYCRS, KEYSPD, KEYFIRE,KARROW,STPOINT,FINLPT
      COMMON NOISE
      LOC (Z=0 , R= 100, W=300)
      HOL (END=END      , HELM AYE=HELM AYE, FIRE AYE=FIRE AYE,
     *     ERROR=ERROR    , ORDERS=ORDERS   )
```

```
  1E  SLJ(N)              SLJ(L+1)
1SAVE SIU1(1REST)         SIL2(1REST)      .SAVE
      SIU3(2REST)         SIL4(2REST)      .INDEX
      SIU5(3REST)         SIL6(3REST)      .REGISTERS
      LDA7(Z+27B)         INA(1)           .PICK UP MONITOR ADDRESS
      SAL(2MSET)          ENI(0)           .
      SAU(L+2)            SAU(L+3)         .SET ADDRESS OF 2M,3M
      ENI2(1)             ENA(-0)          .INITIALIZE IND REG 2
      LDQ2(N)             STQ2(2M)         .SAVE 2M,3M
      STA2(N)             IJP2(L-1)        .USED IN BCD MSG SWITCH
      ENA(ORDERS)         SLJ(1OUT)        .
 1IN  ENA(23B)            SLJ4(Z+11B)      .INPUT PARAMETER TO R BUFFER
      ENA(0)              STA(OUTCL)       .INITIALIZE
      ENA(65B)            ENI(0)           .ENTER AN E
      EQS(R)              SLJ(1CRS)        .IS FIRST LETTER AN E
 1END ENA(45B)            ENI(0)           .  IF SO
      EQS(R+1)            SLJ(1ERR)        .  WAS
      ENA(64B)            ENI(0)           .  END
      EQS(R+2)            SLJ(1ERR)        .  TYPED
      ENA(0)              SLJ(1OUT)        .RESPOND WITH END
2MSET LDA2(2M)            STA2(N)          .RESTORE 2M, 3M
      ISK2(1)             SLJ(L-1)         .
1REST ENI1(N)             ENI2(N)          .RESTORE
2REST ENI3(N)             ENI4(N)          .  INDEX
3REST ENI5(N)             ENI6(N)          .  REGISTERS
      SLJ(1E)             ZRO(0)           .JUMP TO EXIT/ENTRY
  2M  ZRO(0)              ZRO(0)           .2M
  3M  ZRO(0)              ZRO(0)           .3M
```

BLOCK #1

```
 1CRS ENI(0)              ENA(63B)         .ENTER A C
      EQS(R+2)            SLJ(1SPD)        .IF 3RD CHR AN S PROCEED
```

BLOCK #2

```
      ENA(12B)          ENI6(12)          .ENTER A ZERO
      EQS6(R)           SLJ(1ERR)         .LOCATE FINAL ZERO
      INI6(-1)          LDA6(R)           .LOAD 2ND DIGIT
      STA(KEYCRS)       SLJ(1RSPD)        .STORE ORDER AND RESPOND
1SPD  ENI(0)            ENA(22B)          .ENTER AN S
      EQS(R+2)          SLJ(1FIRE)        .IF 3RD CHR A C PROCEED)
      ENA(01B)          ENI(0)            .ENTER A ONE
      EQS(R+7)          SLJ(2SPD)         .IF 1/3 PROCEED
      ENA(1)            STA(KEYSPD)       .STORE SPEED TAG
      SLJ(1RSPD)        ENI(0)            .JUMP TO RESPONSE
2SPD  ENA(02B)          ENI(0)            .ENTER A TWO
      EQS(R+7)          SLJ(3SPD)         .IF  2/3 PROCEED
      ENA(2)            STA(KEYSPD)       .STORE SPEED TAG
      SLJ(1RSPD)        ENI(0)            .JUMP TO RESPONSE
3SPD  ENA(22B)          ENI(0)            .ENTER AN S
      EQS(R+7)          SLJ(4SPD)         .IF STD SPEED PROCEED
      ENA(3)            STA(KEYSPD)       .STORE SPEED TAG
      SLJ(1RSPD)        ENI(0)            .JUMP TO RESPONSE
4SPD  ENA(12B)          ENI(0)            .ENTER A ZERO
      EQS(R+7)          SLJ(1ERR)         .IF ZERO SPEED PROCEED
      ENA(4)            STA(KEYSPD)       .STORE SPEED TAG
1RSPD ENA(HELM AYE)     SLJ(1OUT)         .RESPOND
1FIRE ENA(51B)          ENI(0)            .ENTER A R
      EQS(R+2)          SLJ(1ERR)         .IF 3RD CHR IS R PROCEED
      ENA(66B)          ENI(0)            .CHECK FIRST CHARACTER
      EQS(R)            SLJ(1ERR)         .IF AN F PROCEED
      ENA(1)            STA(KEYFIRE)      .STORE TAG TO SHOOT
      ENA(FIRE AYE)     SLJ(1OUT)         .RESPOND
 1ERR ENA(ERROR)        SLJ(1OUT)
1OUT  STA(OUTCL)        AJP(L+3)          .
      SAU(L+1)          ENI6(0)           .
      LDQ(N)            SLJ(L+2)          .LOAD MESSAGE
      LDQ(END)          ENI6(0)           .LOAD END
      ENA(20B)          STA6(W)           .FILL OUT WRITE BUF
      ISK6(119)         SLJ(L-1)          .WITH SPACES
      ENA(0)            LLS(6)            .PACK W
      STA6(W)           ENI(0)            .
```

BLOCK # 2

BLOCK # 3

```
ISK6(7)          SLJ(L-2)          .
ENA(23B)         SLJ4(Z+12B)       .JUMP TO BCD WRITE
LDA(OUTCL)       AJP(2MSET)        .EXIT
SLJ(1IN)         ZRO(0)            .RESUME PROGRAM
END
```

BLOCK #3

thesJ62
Methods for digital simulation of milita
3 2768 002 10813 6
DUDLEY KNOX LIBRARY